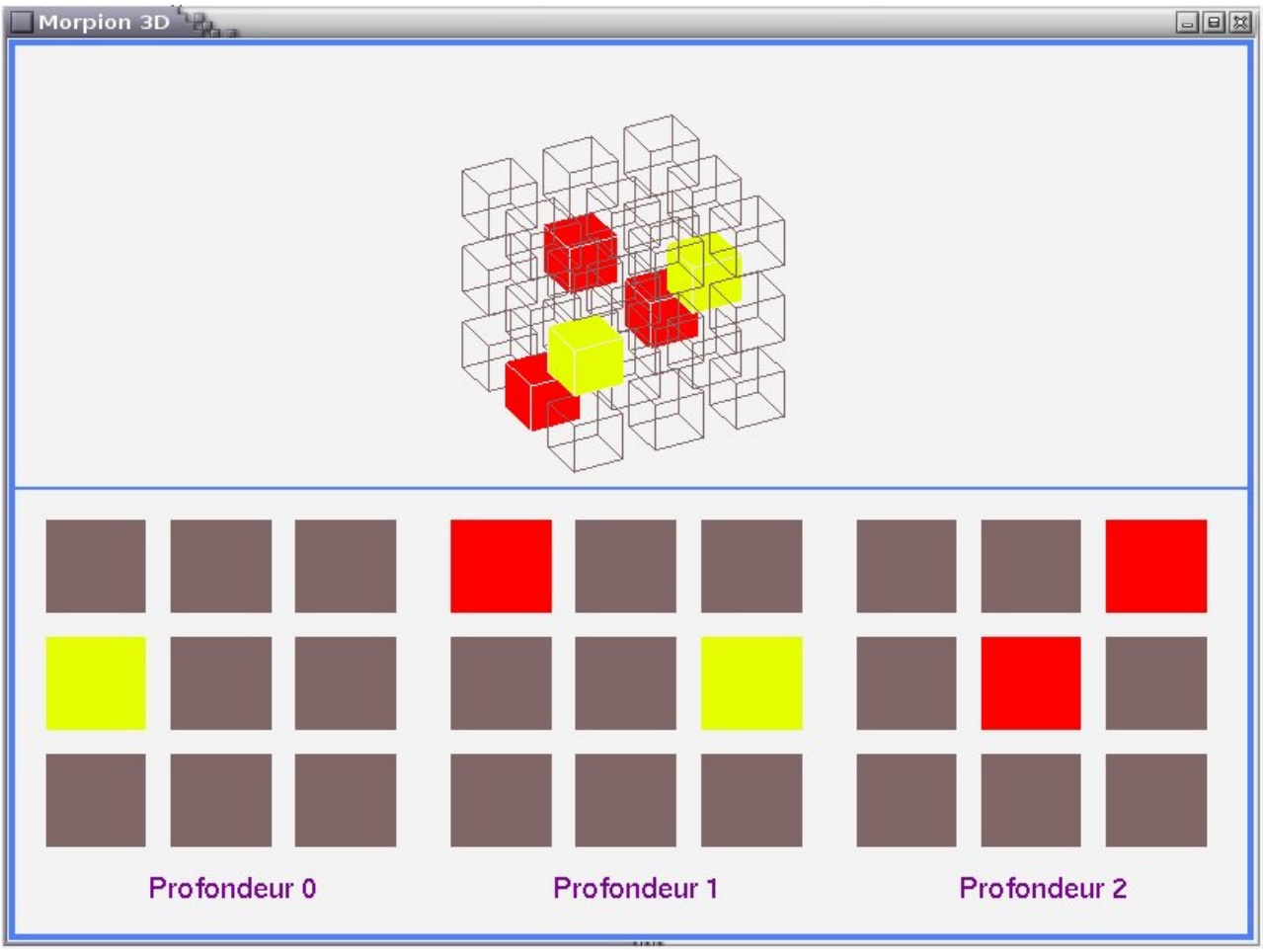
Mini projet C 2004
Morpion 3D

Morpion 3D



BARON Julien
FRANÇOIS Sébastien

Sommaire

I. ETUDE DU PROJET 3
1) Rappel des cahiers des charges 3
2) Bilan des objectifs et des résultats obtenus 4
II. STRATEGIE DE DEVELOPPEMENT 5
1) Organisation 5
1) Répartition du programme en différents fichiers 6
III. MANUEL D'UTILISATION 6
IV. DEFINITION DES STRUCTURES UTILISEES 7
V. LES MACROS UTILISEES 8
1) ContenuTab 8
2) Effacer l'écran 8
VI. L'ALGORITHME DE L'INTELLIGENCE ARTIFICIELLE (IA) 9
1) Etude du fonctionnement de l'IA 9
2) Schéma de fonctionnement de l'algorithme d'intelligence artificielle 10
VII. DETECTION D'UN GAGNANT 11
VIII. EXEMPLES DE JEUX 12
1) L'OpenGL 13
2) En réseau (mode texte) 15
IX. SYSTEME DE SAUVEGARDE ET DE CHARGEMENT 16
1) Sauvegarde 16
2) Chargement d'une partie 16
X. OPENGL 17
XI. RESEAU 24
1) Séquence de connexion 24
2) Protocole utilisé 25
XII. BUGS ET OBSERVATIONS 26
XIII. ANNEXE : LISTING DU CODE 27

I. Etude du projet

1) Rappel des cahiers des charges

Cahier n°1

On souhaite concevoir un jeu de morpion en 3D sur ordinateur, deux joueurs pourront s'y affronter et il est demandé dans un premier temps de gérer la partie, de vérifier que les pions sont correctement placés, et enfin de détecter l'alignement par l'un des joueurs de trois pions entrainant la fin de partie. La troisième dimension sera représentée en affichant successivement les trois plans.

Cahier n°2

On devra pouvoir sauvegarder puis poursuivre une partie.

Cahier n°3

Les joueurs doivent maintenant pouvoir jouer chacun sur son écran, il y aura donc maintenant deux instances du jeu lancées qui devront pouvoir se transmettre des données et permettre de jouer tour à tour.

Cahier n°4

Un nouveau mode de jeu devra permettre de pouvoir jouer seul contre l'ordinateur, une sorte d'intelligence artificielle lui permettra de déterminer où placer ses pions.

2) Bilan des objectifs et des résultats obtenus

Fonctionnalités souhaitées	Résultats obtenus
Cahier des charges n°1 initial - gestion de la partie - vérification des erreurs - détection de la fin de partie	Fonctionnel. La gestion de la partie fonctionne correctement, les joueurs placent tour à tour leur pions, le joueur commençant la partie est tiré au hasard. La vérification des erreurs s'effectue de manière très simple en vérifiant systématiquement le contenu de la case dans laquelle le joueur courant souhaite placer son pion. La détection de la fin de partie s'effectue en plusieurs étapes, nous avons choisit dans un premier temps de ne vérifier que les alignements possibles à partir du dernier pion placé, puis de parcourir les différents plans (X, Y, Z), ensuite le programme teste les diagonales de chaque plan. Il reste enfin à tester les diagonales du cube. Cet ensemble de tests a été développé dans la fonction IsWinner du fichier winner.c
Cahier des charges n°2 Système de sauvegarde	Fonctionnel. La sauvegarde d'une partie s'effectue dans un fichier binaire, il a été choisi d'y sauvegarder le nom du joueur, son identifiant, son pion, le nombre de coup qu'il a joué, et les différentes composantes de sa couleur en OpenGL. Les informations du joueur courant sont sauvegardées en premier. Le fichier porte une extension « .sav ».
Cahier des charges n° 3 Extensions	Fonctionnel. Nous avons opté pour une approche réseau du mode multi-joueur en utilisant des sockets. Le mode de connexion est le suivant : TCP port 9999 L'échange des informations se fait au moyen d'un protocole simple, détaillé plus loin dans la documentation.
Cahier des charges n° 4 Le mode joueur contre ordinateur	Fonctionnel. Dans ce mode l'un des joueurs est contrôlé par un algorithme qui lui permet de déterminer le meilleur coup à jouer pour gagner dans un premier temps, puis pour bloquer l'adversaire si ce n'est pas possible. L'algorithme de cette recherche est détaillé plus loin.

II. Stratégie de développement

1) Organisation

Nous avons défini ensemble plusieurs points concernant la manière de rédiger notre code, de nommer nos fonctions afin d'avoir une écriture plus cohérente.

Les différentes fonctions supplémentaires, à savoir le réseau et l'affichage en OpenGL sont intégrées au code de manière conditionnelle à l'aide de #ifdef

Dans le Makefile :

gcc …-DWITHOPENGL

gcc …-DWITHNETWORK

Permettent d'activer respectivement le code spécifique à l'OpenGL et au réseau.

1) Répartition du programme en différents fichiers

Cahier des charges simples :

coord.c	Manipulation des coordonnées de pions
init.c	Initialisation des données (joueurs, plateau)
jeu.c	Déroulement du jeu (séquentiel en mode texte, événementiel en mode OpenGL)
main.c	Gestion globale du programme
menu.c	Fonctions relatives à l'utilisation des menus
misc.c	Fonctions d'entrée / sortie
screen.c	Affichage texte du plateau
winner.c	Détection de fin de partie et de placement pour l'IA
global.h	En-tête global du programme

Système de sauvegarde :

file.c	Fonctions de sauvegarde et de chargement de parties

Intelligence artificielle :

ia.c	Menu IA et stratégie pour l'IA
random.c	Fonctions relatives à la génération de chiffres aléatoires

OpenGL :

displayglobal_opengl.c	Fonctions gérant le dessin et le redimensionement
displayS1_opengl.c	Fonction de dessin de la fenêtre contenant le cube en 3D
displayS2_opengl.c	Fonction de dessin de la fenêtre contenant les 3 plans
font_opengl.c	Fonctions permettant l'insertion de texte dans la fenêtre
keyboard_opengl.c	Traitement des événements provenant du clavier
mouse_opengl.c	Traitement des événements provenant de la souris
opengl_init.c	Initialisation du système OpenGL et des routines de traitement d'événement
opengl.h	En-tête propre aux fichiers liés à l'OpenGL

Réseau :

client.c	Initialisation, connexion et fermeture d'un client TCP/IP
network.c	Envoi et réception de message par le réseau
server.c	Initialisation, mise en attente et fermeture du serveur TCP/IP
network.h	En-tête propre aux fichiers liés au réseau

III. Manuel d'utilisation

Se reporter au fichier Manuel.txt dans le répertoire du jeu.

IV. Définition des structures utilisées

Les joueurs

Nous devons pouvoir stocker des informations relatives aux joueurs, telles que son nom et sa couleur, mais aussi le nombre de coup qu'il a déjà joué et un identifiant pour pouvoir le repérer sur le tableau de jeu.
Nous avons donc pour cela créé la structure t_joueur dont voici le prototype :

```
typedef struct t_joueur
{
      char *nom ;       //Nom du joueur
      char nb ;               //Identifiant du joueur
      t_color couleur ;       //Couleur du pion du joueur
      unsgned char nbcoup ;   //Nombre de coup joué par le joueur
} t_joueur ;.
```

Les couleurs

Dans cette structure, on voit apparaître une nouvelle structure, la structure t_color. Cette structure est dédiée à l'affichage graphique, elle permet d'enregistrer la couleur d'un pion dans le mode RGB. Il faut cependant noter que le RGB est quelque peu modifié. Ce ne sont plus des entiers de 0 à 255, mais des flottants de 0.0 à 1.0.

```
typedef struct t_color
{
      float red, green, blue ;      //Composante RGB d'une couleur
} t_color ;
```

Les coordonnées

Pour faciliter les passages de coordonnées entre les fonctions, nous avons également créé une structure COORD qui stockera les abscisses, ordonnées et hauteurs d'une case ou d'un pion.

```
typedef struct COORD
{
      unsigned char x, y, z ; //Abscisse, ordonnée et hauteur
} COORD ;
```

Ces trois structures seront utilisées tout au long du jeu pour faciliter le passage en argument à certaine fonction, et dans un souci de meilleure lisibilité du code.

Les menus

Nous avons une dernière structure quelque peu à part pour définir nos menus : la structure t_menu.

```
typedef struct t_menu
{
      char *titre ;                       //Titre du menu
      char *tabmenu[LENMENUMAX] ;   //Les différents menus proposés
      int lenmenu ;                       //Le nombre de menu proposé
}
```

La valeur LENMENUMAX est définie en #define et correspond au nombre maximum de caractères que peut contenir un élément du menu.

Cette dernière structure est utilisée pour rendre plus générales certaines fonctions d'affichage, mais nous verrons ceci plus en détails ultérieurement.

V. Les macros utilisées

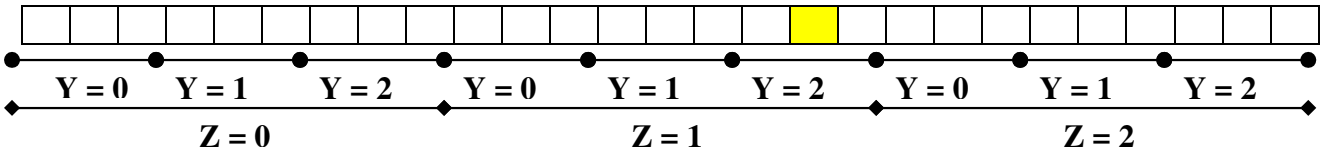
1) ContenuTab

La macro ContenuTab permet d'accéder à une case du plateau de jeu. Sa définition est la suivante :

```
#define ContenuTab(tab, x, y, z)    *(tab + (z * NBY * NBX) + (y * NBX) + x)
```

Il est à noter que notre plateau de jeu est défini de la sorte : char plateau[NBX][NBY][NBZ]

Ci-dessous une représentation de notre plateau de jeu en mémoire :



Ce tableau permet d'expliquer à lui seul la macro ContenuTab. Si l'on veut aller se positionner sur la case de coordonnées (1, 2, 1), il faut alors aller à la case 16 du plateau inscrit en mémoire, ce qui correspond bien à 1 * 3 * 3 +2 * 3 + 1 = 16, en admettant que l'on est bien paramétré NBX, NBY et NBZ à 3.

2) Effacer l'écran

Cette macro permet d'effacer tout le texte précédemment écrit dans la console.

```
#ifdef LINUX
      #define EraseScreen()   system("clear" );
#else
      #define EraseScreen()   system("cls");
#endif
```

Cette macro se fait en deux étapes. On vérifie tout d'abord le type du système d'exploitation par une variable LINUX définie dans le Makefile. Ainsi, pour quelques rares fonctions, nous écrivons deux versions.

Dans le cas de notre macro, si nous travaillons sous linux, chaque fois que dans le code nous aurons la fonction EraseScreen(), elle sera remplacée par system("clear"), et sous Windows par system("cls").

VI. L'algorithme de l'intelligence artificielle (IA)

1) Etude du fonctionnement de l'IA

Après analyse de nombreux exemples, il s'est avéré que la case centrale du cube est primordiale pour la victoire.

Premier tour de l'IA :
• L'IA commence donc par tester si la case centrale du plateau est prise. Si ce n'est pas le cas, elle joue sur cette case. Au deuxième tour de jeu de l'IA ou du joueur, cette case est donc prise.
Deuxième tour de l'IA si elle a commencé :
• Si l'IA a commencé, elle a joué au centre et l'adversaire a joué sur une des 26 autres cases. L'IA joue alors sur une des cases avoisinant le pion adverse, en faisant varier une seule coordonnée (x, y, ou z). Le joueur adverse est alors obligé de contrer l'IA qui a 2 pions alignés et ne peut en contrant l'IA aligner 2 pions et donc avoir une possibilité de gagner au tour suivant.
Tous les autres cas :
• Pour les autres tours, le principe de l'IA est simple. Elle regarde si elle peut gagner. Si elle a plusieurs possibilités de gagner, elle renvoie au hasard une des positions sur lesquelles elle peut gagner.
• Si ce n'est pas le cas, elle regarde si l'adversaire peut gagner, et de la même façon joue sur une des cases au hasard où l'adversaire peut gagner.
• Autrement, si ni l'IA, ni le joueur ne peuvent gagner, l'IA regarde si en jouant sur une case, elle ne peut pas aligner deux fois deux pions pour avoir deux possibilités de gagner au tour suivant, et l'IA joue alors sur une de ces cases.
• Si ce n'est toujours pas le cas, l'IA regarde si le joueur peut aligner deux fois deux pions, et joue sur une des cases où cela est possible.
• Dans tous les autres cas de figures (je doute qu'il y en est beaucoup à moins d'en faire vraiment exprès, et encore), l'IA joue sur une des cases qui lui offre le plus de possibilités d'alignements sachant que la case du milieu est prise.

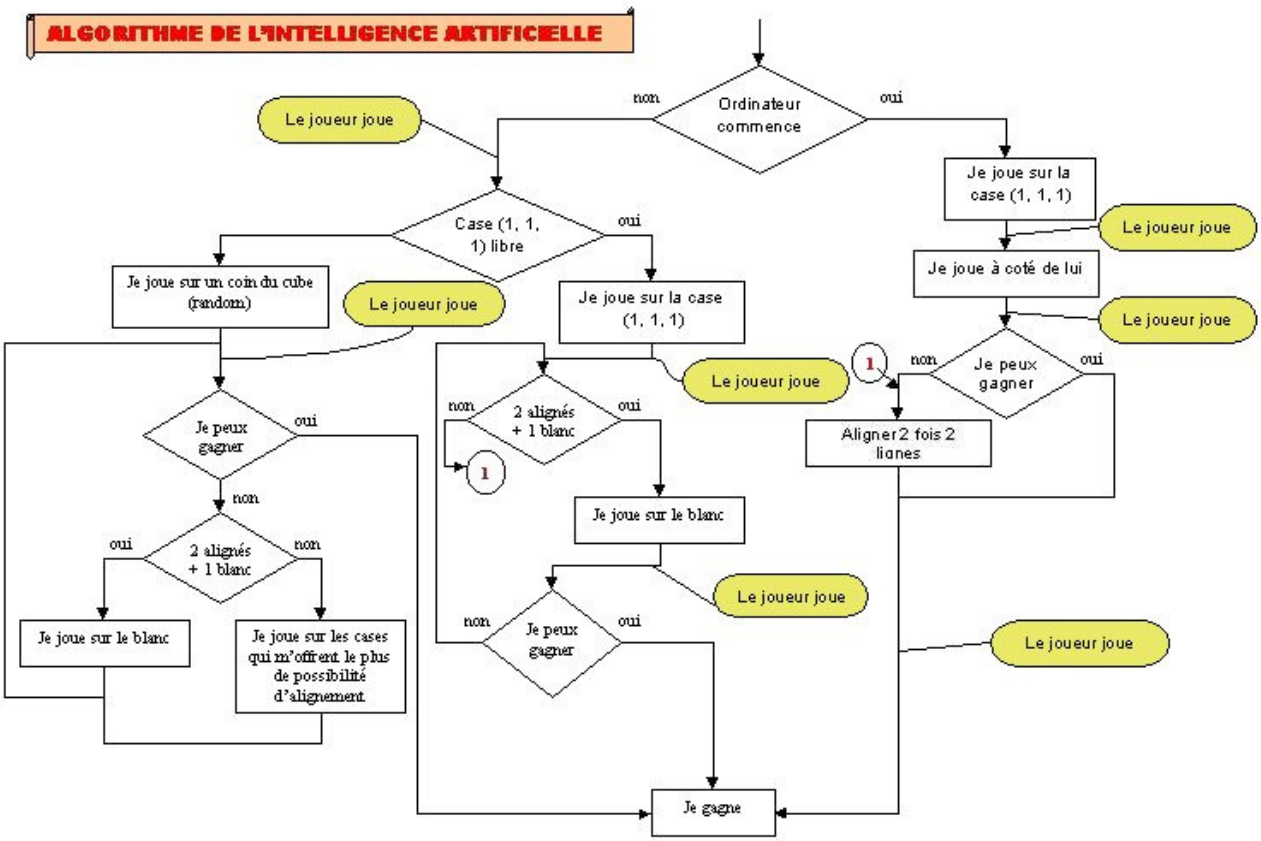
Amélioration de l'IA

Pour améliorer cette IA, il aurait été possible de ne pas renvoyer une position au hasard, mais de spécifier par l'intermédiaire de flags les cases les plus avantageuses. Prenons par exemple le cas ou l'IA ne peut pas gagner mais que le joueur adverse le peut. Pourquoi ne pas jouer sur une case où l'adversaire peut gagner et jouer en même temps sur une case qui nous permet d'aligner deux fois deux pions ? De même dans l'éventualité où l'on contre une possibilité de victoire, ne devrait-on pas jouer sur la case la plus visible, comme un alignement en droite horizontale ou verticale plutôt que de bloquer une victoire par diagonale que le joueur risque de ne pas voir ?

Statistiques sur l'IA

Après étude, il s'avère que si l'IA commence à jouer, elle ne peut que gagner en 4 coups maximums. De même, si le joueur commence ailleurs qu'en la case centrale, l'IA ne peut que gagner, le temps dépend de la façon de jouer du joueur. Cette dernière affirmation n'est cependant pas prouvée, nous n'avons pas testé tous les cas, mais dans les multiples phases de tests effectués, nous n'avons jamais pu vaincre l'IA si nous ne jouions pas sur la case centrale.

2) Schéma de fonctionnement de l'algorithme d'intelligence artificielle



VII. Détection d'un gagnant

La détection se fait au travers d'une seule et unique fonction dont voici le prototype:

```
unsigned char IsWinner(char *tab, t_joueur joueur, COORD lastplay);
```

Cette fonction doit être simple et rapide à exécuter puisqu'elle est appelée à chaque tour de jeu, et de nombreuses fois pour trouver la position du pion joué par l'IA.

On a donc remarqué qu'il n'était pas nécessaire de parcourir le tableau en entier à chaque fois pour déterminer si un joueur avait ou non gagné. Il suffit de regarder si avec le dernier pion joué par le joueur, un alignement de NBPIONSALIGN était réalisé. Pour cela, il faut tester dans toutes les directions possibles (horizontales, verticales et diagonales) qu'un alignement n'est pas réalisé.
Pour mieux comprendre cette fonction, nous allons étudier une partie du code de la fonction:

```
char i, nb;
unsigned char cmpt;

cmpt = 0;
nb = 0;
for (i = -(NBPIONSALIGN - 1); i < NBPIONSALIGN - 1; i++)
{
	if( (lastplay.x + i) < 0 || (lastplay.x + i) >= NBX)
		continue;
	if (ContenuTab(tab, lastplay.x + i, lastplay.y, lastplay.z) ==
joueur.nb)
		nb++;
	else
		nb = 0;
	if (nb >= NBPIONSALIGN)
	{
		cmpt++;
		break;
	}
}
…
```

Cette boucle for teste un alignement suivant les x. A chaque pion trouvé on incrémente nb jusqu'à ce que nb soit supérieure ou égale à NBPIONSALIGN.

A ce moment la, si on a atteint le nombre de pions à aligner, on incrémente cmpt. cmpt représente le nombre de fois où l'on a gagné en jouant sur cette case (le nombre d'alignement effectué).

On remarque le test de validité de l'équation au début de la boucle. Ce test est essentiel pour ne pas déborder du tableau ou tester sur des cases qui ne doivent pas l'être.

Pour vérifier qu'il y a un gagnant il ne reste plus qu'à réitérer cette boucle for sur chaque alignement possible (selon les x, y ou z, ou encore selon les plans xy, xz ou yz, ou encore selon les diagonales particulières du cube)

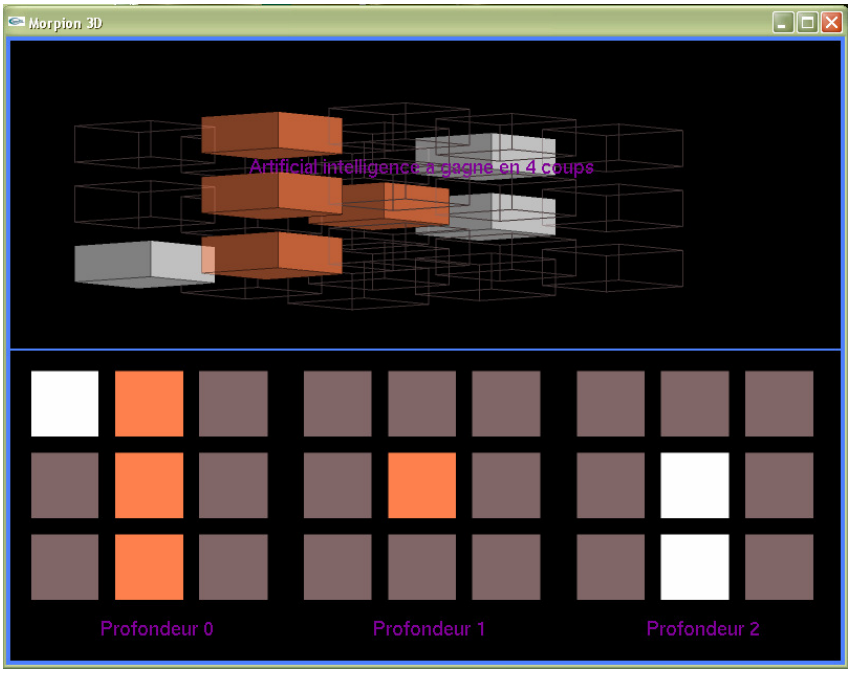
VIII. Exemples de jeux

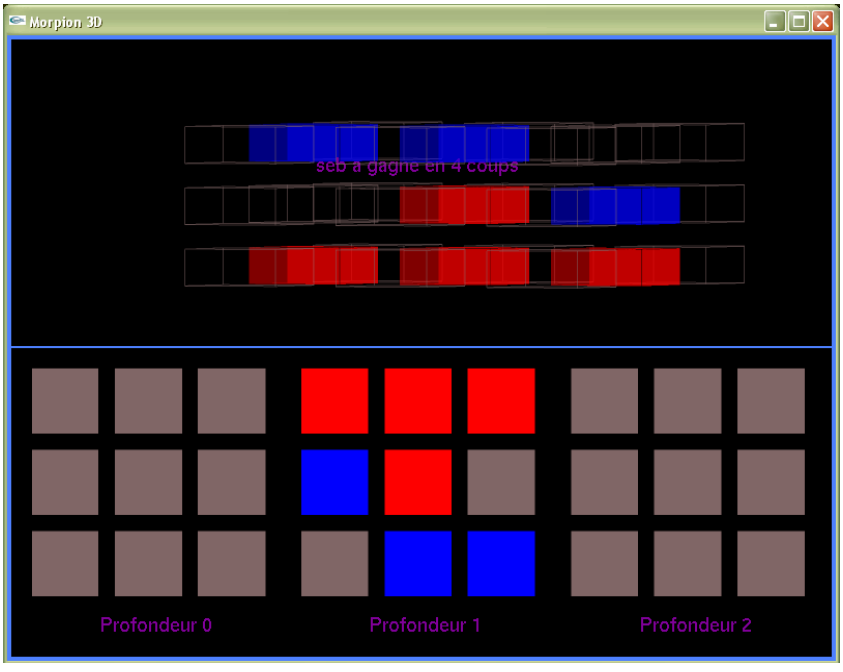
Ci-dessous deux exemples de parties finies avec un vainqueur.

Nous avons essayé de faire un match nul, mais n'y sommes pas parvenus !

L'affichage d'un vainqueur aurait pu se faire plus joliment mais le manque d'enthousiasme face à un objectif purement esthétique a eu raison de nous.

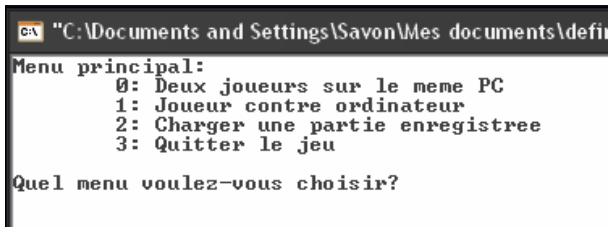
L'important côté graphique était la gestion de la librairie OpenGL, problème abordé plus loin dans ce compte-rendu.



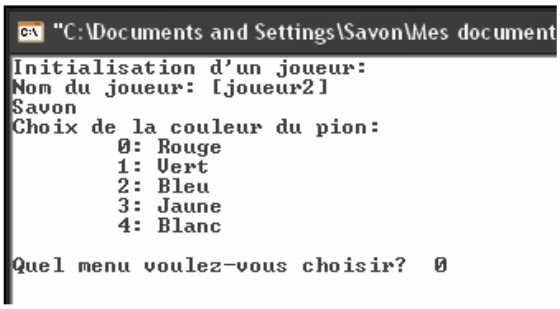


1) L'OpenGL

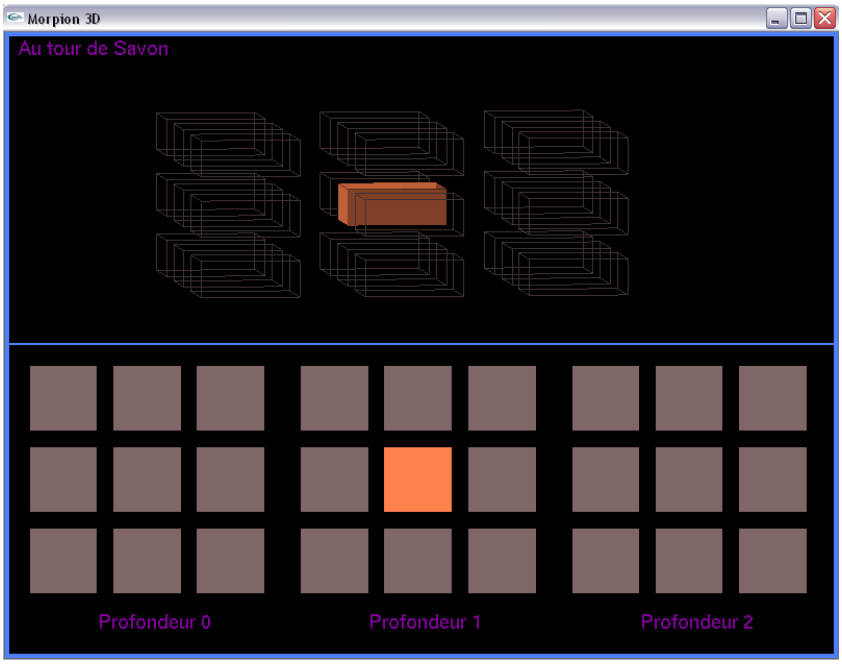
Tout d'abord, choisir un mode de jeu (pour des raisons de commodité, nous vous montrerons un jeu avec l'intelligence artificielle que nous avons implémentée!)
Nous nous excusons pour la qualité des images, mais nous avons préféré ne pas prendre trop de place sur les feuilles pour présenter uniquement des captures d'écrans!
Voici le déroulement du jeu:


"C:\Documents and Settings\Savon\Mes documents\defi
Menu principal:
0: Deux joueurs sur le meme PC
1: Joueur contre ordinateur
2: Charger une partie enregistree
3: Quitter le jeu
Quel menu voulez-vous choisir?

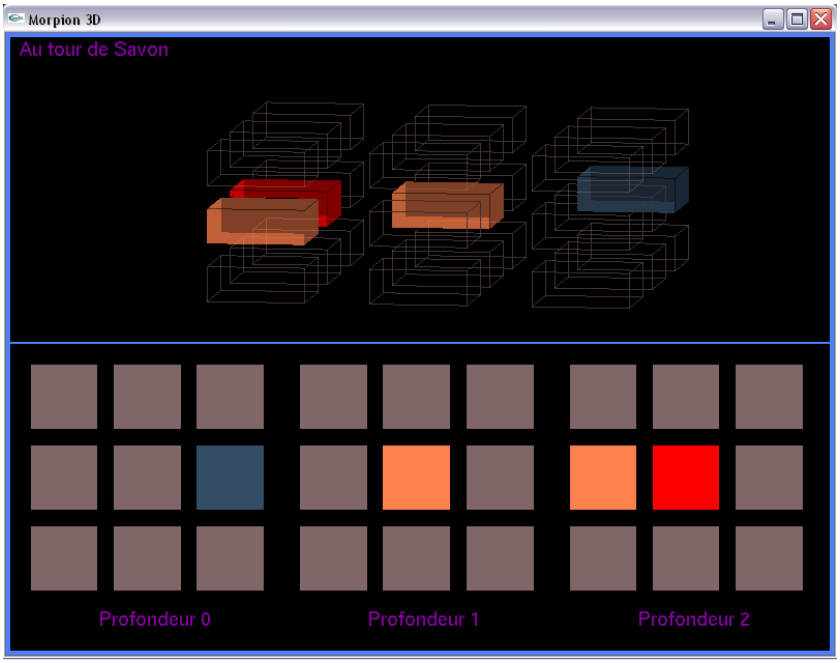
Choix du jeu (menu principal)



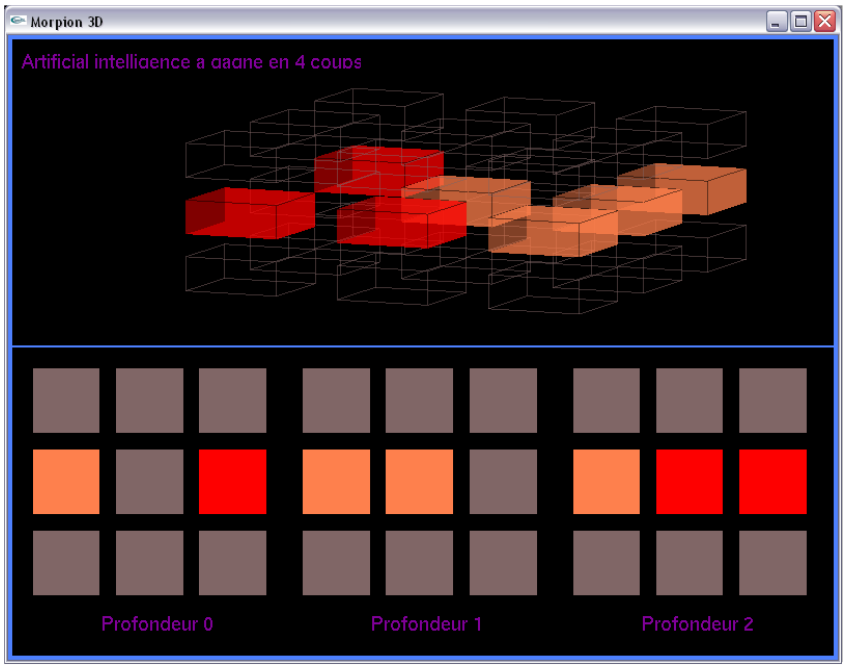
Initialisation des données concernant le joueur



Début du jeu (le rose saumon est la couleur de l'IA)

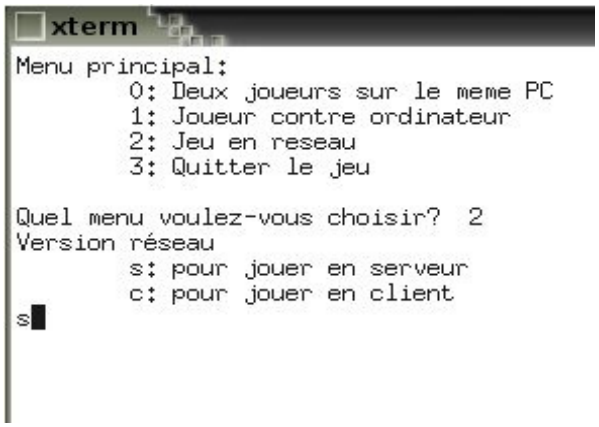


Au cours du jeu
(on remarque le pion rouge du joueur et le pion bleu clair indiquant une case sur laquelle on test le pion, ce pion bleu clair n'est pas définitif)

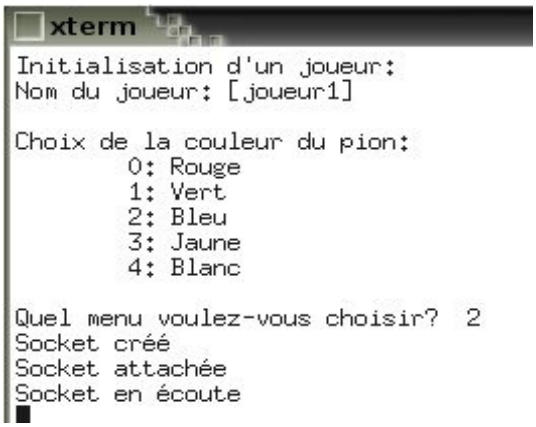


Comme toujours, l'intelligence artificielle nous a vaincus en quatre tours, sniff!

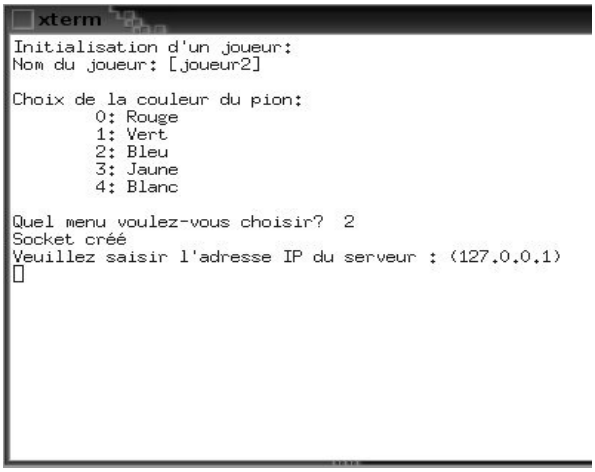
2) En réseau (mode texte)



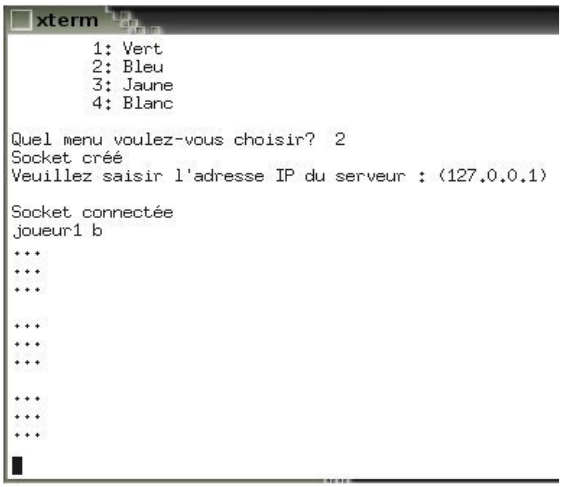
Choix du rôle de serveur réseau



Serveur en attente



Saisie de l'ip du serveur du coté du client



Connexion établie

Du coté du perdant !

xterm
Morpion 3D realise par Savon et Seb
Pour quitter la partie tapez la lettre 'Q'
lettre 'S'
joueur1 a gagne en 3 coups
Appuyez sur ENTREE pour continuer...

Vue du gagnant !

IX. Systeme de sauvegarde et de chargement

1) Sauvegarde

Pour sauvegarder, on appelle la fonction :

```
int SaveGame(char* tab, t_joueur joueur1, t_joueur joueur2) ;
```

Cette fonction sauvegarde le jeu dans un fichier du type *.sav (appel à la fonction CreateFile)

La sauvegarde comprend 4 étapes :

1.-Ecriture dans le fichier d'un en-tete qui permet de vérifier la validité du fichier. Si le fichier ne contient pas cet en-tete, il ne s'agit pas d'un fichier de sauvegarde du jeu.

2.-Ecriture du tableau de jeu en une seule ligne, sans espace. Comme dans le tableau de jeu, on écrit le numéro du joueur (1, 2, 3, …), cette ligne est remplie de blanc ou de caractère étrange. Ce qui limite un peu la tricherie aussi !!!!

3.-Ecriture des informations concernant le joueur qui doit jouer au prochain chargement du jeu.On écrit dans l'ordre et en sautant une ligne à chaque fois le nom du joueur, le numéro du joueur, les composantes RGB de la couleur choisie par le joueur et le nombre de coups joués par le joueur.

4.-Ecriture de la même façon des informations concernant le second joueur.

2) Chargement d'une partie

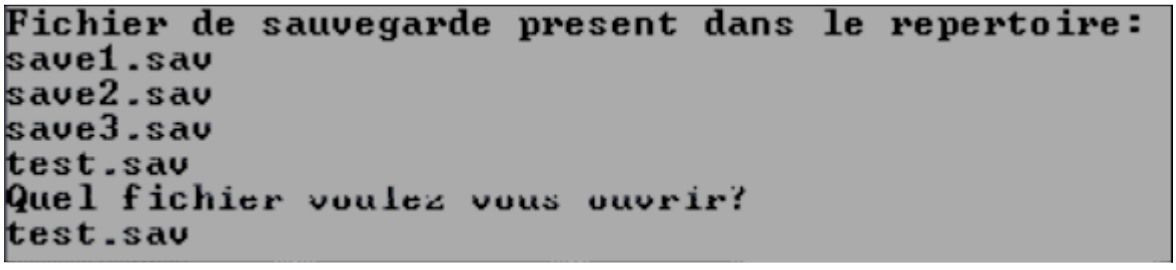
Le chargement d'une partie à partir de fichier de sauvegarde se fait dans l'ordre inverse. On fait appel pour cela à la fonction :

```
int LoadGame(char* tab, t_joueur *joueur1, t_joueur *joueur2)
```

Cette fonction affiche dans un premier temps tous les fichiers portant l'extension *.sav et demande à l'utilisateur de choisir le fichier qu'il souhaite charger.

Cette phase de chargement s'effectue en 4 parties :

1.-Vérification de la validité du fichier (lecture et comparaison de l'en-tete)
2.-Initialisation du tableau à partir des données contenues dans le fichier.
3.-Initialisation du joueur1 (c'est le joueur qui jouera en premier)
4.-Initialisation du joueur2.



Exemple d'un fichier de sauvegarde

```
FICHIER DE SAUVEGARDE DU MORPION 3D
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
joueur1
□
y
2
0.900000:1.000000:0.000000
joueur2
□
b
2
0.000000:0.000000:1.000000
```

X. OpenGL

Pour apprendre à utiliser la librairie OpenGl, nous avons utilisé le tutorial suivant :

http://www.lighthouse3d.com/opengl/glut/

Présentation d'OpenGL

OpenGL propose un ensemble de fonctions pour la synthèse d'images 3D (plus accessoirement 2D). Différents modes de rendu sont possibles : fil de fer, faces pleines avec ou sans modèle d'éclairage, textures, etc. De nombreuses implantations de l'api exploitent les capacités matérielles de la plate-forme (logiciels pilotes des cartes graphiques).

L'architecture d'OpenGL est du type "machines abstraites", avec des variables d'états (variables statiques) accessibles uniquement par des fonctions d'OpenGL, et dont les valeurs persistent jusqu'à la modification suivante.

Prototypes des fonctions

Les fonctions OpenGL commencent par gl. Certaines sont suffixées par un indicatif du nombre et du type d'arguments :

```
gl...{234}{fidbsu}[v]();
```

Capacités de rendu

Le rendu effectué est simple, il s'appuie sur la projection dans l'espace image de polygones plans. Cette technique est rapide mais ne permet pas de rendre les interactions lumineuses complexes (à l'inverse de la technique du lancé de rayons).

Ce qu'OpenGL fait :
- Affichage de polygones plans (toutes les primitives sont transformées en polygones).
- Projection perspective et orthographique.
- Gestion des transformations et des normales.
- Gestion des faces cachées (élimination par l'orientation + technique du z-buffer).
- Couleurs RVB, mélange (composante alpha).
- Textures.
- Éclairage. Implémente les modèles de lumière Gouraud et Phong.
- Simulation du brouillard.

Ce qu'OpenGL ne fait pas :
- Lancer de rayons et les possibilités associées :
- Projection d'ombres et simulation de la réflexion / réfraction.
- Gestion d'un format de fichier 3D, d'un format de fichier image.
- Gestion des fenêtres et interaction avec l'utilisateur (clavier, souris).

On utilisera GLUT (GL Utility Toolkit) qui facilite la création des fenêtres et la programmation des interactions par clavier et souris.

GLUT et GLU offrent également certaines fonctions de plus haut niveau : primitives géométriques de base, aide au placement des caméras, aide au texturage, etc.

Plans mémoires

OpenGL organise les données qu'il manipule en plans mémoires (ou buffer)

– Mémoire écran ou Color Buffer : contient les pixels de l'image rendue. On peut également effectuer le rendu dans des plans auxiliaires, utilisés ultérieurement.

– Tampon de profondeur ou z-buffer : contient l'information de profondeur associée à chaque pixel (distance du plus proche objet).

– plan de stencil : calque fixe devant l'écran.

– plan d'accumulation : utilisé pour des effets complexes comme l'anti-crénelage ou la simulation de profondeur de champ.

Modélisation de scène

On présente ici les bases pour modéliser géométriquement une scène, c'est à dire placer des objets, définir une caméra et l'orienter convenablement.

Quelques primitives géométriques de base

Dans la pratique on construit les objets géométriques par assemblage de primitives de base ou comme ensemble de polygones. Les primitives de bases sont elles-mêmes constituées de polygones.

Note : Les primitives ainsi définies sont placées et orientées relativement à un repère local: par exemple la sphère a son centre en l'origine du repère. On verra plus loin comment déplacer et orienter ce repère dans le repère monde.

- `void glutWireCube(GLdouble taillecote);`
- `void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);`

- `void glutWireTorus(GLdoublerayontube, GLdouble rayontore, GLint slice, GLint stacks);`

Matrices de transformations et de projection

Un polygone (par extension toute primitive graphique) subit une série de transformations avant son affichage à l'écran. Cette suite de traitements est appelée le pipeline graphique.

OpenGL utilise 2 matrices 4x4 appelées `GL_PROJECTION` et `GL_MODELVIEW` pour modéliser ces transformations. Elles agissent sur les sommets et les normales définis en coordonnées homogènes. Après "passage" par ces deux matrices, la division perspective (par la quatrième coordonnée) intervient, les coordonnées sont normalisées puis rapportées au repère écran (voir figure 1).

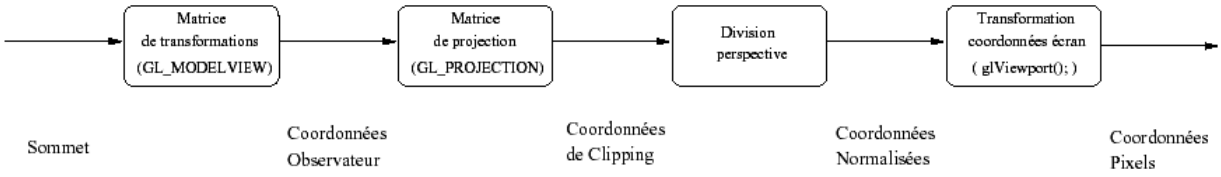


FIG. 1 – Le pipeline graphique d'OpenGL

GL_PROJECTION permet de modéliser la projection effectuée par la caméra. On la sélectionne avec glMatrixMode(GL_PROJECTION), on peut la paramétrer avec les fonctions suivantes ou bien manuellement:

- `gluPerspective(angle, rapportLsurH, znear, zfar);`
- `glFrustum(left, right, bottom, up, near, far);`
- `glOrtho(left, right, bottom, up, near, far);`

`GL_MODELVIEW` permet de modéliser les transformations géométriques à appliquer à la scène, que ce soit pour déplacer les objets ou la caméra (dual). On la sélectionne avec `glMatrixMode(GL_MODELVIEW)`

On peut la paramétrer avec les fonctions suivantes ou bien manuellement :

- `glTranslatef(x, y, z);`
- `glRotatef(angle, x, y, z);`
- `glScalef(x, y, z);`

Pour faciliter le placement de la caméra :

```
void gluLookAt(
      GLdouble eyeX , GLdouble eyeY , GLdouble eyeZ,
      GLdouble centerX , GLdouble centerY , GLdouble centerZ ,
      GLdouble upX, GLdouble upY , GLdouble upZ
);
```

Ces fonctions modifient la matrice `GL_MODELVIEW` en cours en la multipliant à droite par la matrice correspondant à la transformation demandée.

`glLoadIdentity()` permet de charger une matrice identité dans la matrice en cours.

Les piles de matrices

Il existe en fait une pile de matrices pour manipuler `GL_MODELVIEW` (de même pour `GL_PROJECTION`).

La matrice utilisée à un instant donné est celle située au sommet de la pile. La manipulation des piles de matrices est réalisée à l'aide de `glPopMatrix()` et `glPushMatrix()` (qui agissent sur la pile en cours ie `GL_PROJECTION` ou `GL_MODELVIEW`).

`glPushMatrix()` copie la matrice active vers le sommet de la pile. Cela permet de sauvegarder la matrice en cours.

glPopMatrix() restaure la matrice située au sommet de la pile. Ceci permet de revenir en arrière dans une configuration sauvée par glPushMatrix().

Méthode pour placer les objets

Les objets géométriques sont définis dans un repère lié à l'objet. Placer l'objet dans la scène, c'est amener le centre du repère lié à l'objet à l'endroit voulu, et l'orienter correctement. Les transformations appliquées au repère objet sont aussi appliquées à l'objet.

Initialement repère objet et repère monde sont confondus (`GL_MODELVIEW` = identité). On écrit une suite de transformations afin d'amener le repère objet à l'endroit voulu dans le repère monde. La transformation i doit être définie relativement au repère objet dans l'état transformé par la suite des (i - 1) ièmes précédentes transformations. L'ordre d'écriture du code est celui des transformations successives.

Méthode pour placer la caméra

Rappelons que les placements des objets et de la caméra sont duals, puisque qu'on agit sur la même matrice `GL_MODELVIEW`. Un déplacement de la caméra s'interprète comme une transformation à appliquer à *tous les objets de la scène*, par conséquent on écrit la transformation de l'espace correspondante en premier dans le code. Une fois installée, la transformation caméra voulue doit être sauvée (avec `glPushMatrix())` si l'on souhaite placer plusieurs objets relativement au repère monde sans recalculer la transformation à chaque fois.

La caméra est placée au centre du repère monde et regarde vers les Z négatifs.

On retiendra les 2 approches suivantes :

– Tourner autour d'un objet : translation puis rotation.
– Regarder la scène depuis un point donné : rotation puis translation.

Les couleurs

OpenGL est capable de travailler avec différentes représentations des couleurs. La plus facile à utiliser est le mode RGB, dans lequel chacune des 3 composantes rouge, verte et bleue est codée sur 8 bits. Un appel à la fonction `glColor3f()` permet de définir la couleur pour les tracés suivants, jusqu'à un prochain appel pour choisir une autre couleur. Le prototype de la fonction est le suivant :

```
void glColor3f (GLfloat r, GLfloat g, GLfloat b);
```

`r`, `g` et `b` sont les proportions de chaque composante (valeur entre 0 et 1).

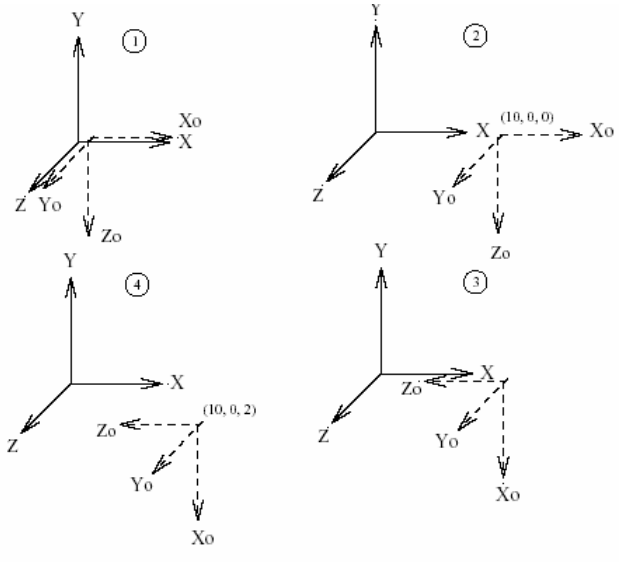
Les fonctions en 4f permettent de jouer sur la transparence.

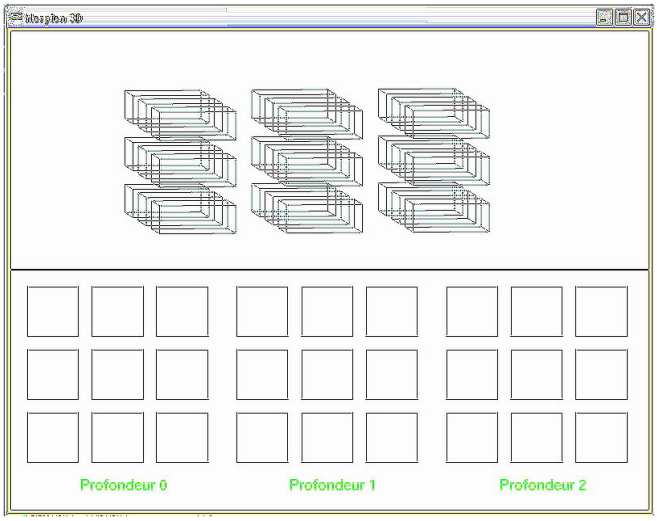
Exemple d'une transformation géométrique

Soit la séquence :

```
glRotatef (90, 1, 0, 0);      // (1) Rotation autour de X de 90
glTranslatef (10, 0, 0);      // (2) Translation de 10 selon X
glRotatef (-90, 0, 1, 0);     // (3) Rotation de -90 autour de Y
glTranslatef (0, 2, 0); // (4) Translation de 2 selon Y
```

L'interpretation graphique dans le sens de lecture du code est présentée ci-dessous. Elle correspond à une transformation du repère local à l'objet.





Description de la fenêtre :

Nous créons une fenêtre de dimension de base 800*600 pixels. Nous subdivisons ensuite cette fenêtre en deux sous fenêtres dans lesquelles nous affichons respectivement un cube composé de 27 petits cubes en 3D qui pourra tourner selon l'utilisation de la souris, et dans la seconde fenêtre, nous affichons ce même cube coupé en 3 plans.

La qualité de l'affichage est bien meilleure, mais un traitement de l'image a été appliqué pour limiter l'utilisation du noir, le fond de la fenêtre étant de cette couleur.

Diverses fonctions ont donc dues être mises en place pour gérer le redimensionnement de la fenêtre et la réactualisation de cette fenêtre. Pour charger ces fonctions dans l'environnement OpenGL, il faut faire appel à des fonctions qui se chargent elles-mêmes de l'intégrer. Ces fonctions ont toutes des prototypes imposés, que nous sommes obligés de suivre. Toutes les données que nous voulions donc passer en paramètre à nos fonctions d'affichages doivent alors être mises en variables globales.

Le clavier

Une interaction événementielle avec le clavier a été mise en place.

Ainsi, dans le cas où les joueurs veulent sauvegarder la partie, ils n'ont qu'à appuyer sur la touche 'S' qui ferme automatiquement la fenêtre OpenGL, et lance la fonction de sauvegarde.

Deux fonctions sont nécessaires à la bonne gestion du clavier. Une fonction regarde les touches classiques du clavier, et une autre qui regarde les touches plus spécifiques d'un clavier comme les douze touches F1…F12.

Ainsi, pour fermer le programme, il est possible d'utiliser la combinaison de touches ALT+F4.

De même, si un joueur a gagné, l'appui sur n'importe quelle touche nous ramènera au menu principal.

La souris

Outil de base de notre application, la souris est utilisée pour faire tourner le cube en 3D et ainsi pouvoir voir le cube sous tous les angles.

Elle est aussi utilisée pour valider une case. Le joueur, dans la deuxième fenêtre du jeu, choisit la case sur laquelle il veut jouer et clique pour valider cette case. Par un traitement de la position du clic, on arrive à retrouver les coordonnées de la case dans laquelle il a joué, et ainsi à insérer son pion.

Mais là encore, deux modes de saisie s'offrent au joueurs. Si le joueur clique sur une case avec le bouton droit de la souris, le programme enregistre le coup au titre d'un essai, et affiche la case sélectionnée sous une autre couleur pour permettre au joueur de vérifier qu'il s'agit bien de la case où il voulait jouer. (Et oui, la 3D joue de si vilains tours à nos yeux !). Tandis que si il clique avec le bouton gauche de la souris sur une case, son coup est enregistré, et on passe au joueur suivant.

Le cube et sa rotation

On calcule les déplacements horizontaux et verticaux de la souris lorsque le clique gauche est enfoncé et que la souris se situe dans la fenêtre du cube.

En fonction de la valeur de ces déplacements, on fait appel à la fonction glRotatef() qui effectue une opération de rotation sur la matrice représentant notre cube, et on affiche cette matrice à l'écran.

On pourra remarquer que le cube n'est pas réellement cube, puisque la dimension de ses côtés dépend de la dimension de la hauteur et de la largeur de la fenêtre. On dira donc plutôt qu'il s'agit d'un pavé.

Le dessin d'un petit cube du gros cube se fait en six étapes. A chacune de ces étapes, nous dessinons une face du cube. Au lieu de procéder de la sorte, nous aurions pu utiliser les possibilités offertes par GLUT, mais n'y ayant pas pensé au début, nous ne sommes pas revenus sur cette partie du code.

Les difficultés rencontrées avec OpenGL

De nombreuses difficultés ont été rencontrées durant ce mini projet.

La plus grosse de ces difficultés, l'utilisation d'OpenGL dans son ensemble. En effet, nous avions commencé par réaliser une application en mode console, donc une programmation de type itérative. Puis lorsque nous sommes passés à OpenGL, alors que nous pensions réutiliser le code préalablement tapé, nous nous sommes rendus compte qu'il fallait utiliser une optique complètement différente, et programmer d'un point de vue évènementiel.

Ensuite, nous nous sommes heurtés évidemment à plusieurs pièges. Tout d'abord avec la gestion d'une fenêtre. Nous étions habitués à travailler sur des fenêtres dont l'origine est en haut à gauche, mais là, ô surprise, l'origine est au centre, et les axes sont gradués de -1 à 1. Il a donc fallu trouver une méthode pour changer la position de cette origine et la valeur des axes.

Un autre problème que nous avons déjà abordé est le prototypage imposé de la majorité des fonctions. Ceci nous a encore obligé à revoir une bonne partie du code.

Lorsque le jeu fonctionnait en mode OpenGL, nous avons alors travaillé sur l'aspect graphisme, en ajoutant des effets de transparence et de lumière. Mais là, des problèmes d'ordre de la compatibilité windows-linux nous ont posé quelques soucis. En effet, avec X-window sous linux, le logiciel refusait de se lancer. Ceci était dû à l'utilisation de buffer de 32 bits.

Un autre problème était également présent lorsqu'une partie était finie. Nous souhaitions revenir au menu présent en console. Et le problème, la seule méthode apparente pour quitter la boucle évènementielle d'OpenGL : un appel à la fonction exit(). C'est-à-dire que nous fermions le programme sans pour autant revenir au menu principal. Comment faire ? Nous avons détruit la fenêtre OpenGL (mais sa boucle événementielle tourne encore) puis nous appelons le main. Ce n'est pas très beau, mais cela semble fonctionner. Et de ce main, où nous recréons la fenêtre OpenGL qui récupère sa boucle évènementielle. Nous quittons le programme par un appel à exit() qui détruira la boucle évènementielle restante.

XI. Réseau

Notre référence a été un TP réseau « programmation socket sous windows »de DUT qu'il a fallu croiser avec différents man pour adapter sous Linux.

Un socket est une interface réseau spécifique, ici nous l'utilisons en TCP, un protocole connecté et fiable (il permet de s'assurer que le paquet envoyé est bien reçu). Nous échangerons dessus de manière bidirectionnelle les informations nécessaires au déroulement du jeu.

Le port choisi (9999) ne correspond à aucun port standard de protocole à notre connaissance.

1) Séquence de connexion

Serveur	Client
socket (ouverture d'un socket)	socket (ouverture d'un socket)
bind (attachement du socket à une interface de la machine)	bind (attachement du socket à une interface de la machine)
listen (mise en attente)	
accept (accepte une connexion distante)	connect (transmet une séquence d'établissement de connexion)
recv (reçoit un message sur le socket)	send (reçoit un message sur le socket)
send (reçoit un message sur le socket)	recv (reçoit un message sur le socket)
close (ferme le socket et réinitialise la connexion)	close (ferme le socket et réinitialise la connexion)

2) Protocole utilisé

Nous avons mis en place un protocole très simple, basé sur un langage « humain ».

Dans cet exemple le client aura pour nom « joueurclient », le pion 'c', le serveur sera « joueurserver » et aura pour pion 's'.

Une fois la connexion établie, le serveur se met en réception, et le client transmet son nom et son pion :

Client->Serveur : Hello, I am joueurclient (c).
Serveur->Client : Hello, I am joueurserveur (s).

--- Le serveur commence et doit entrer les coordonnées de son premier pion (x :1 y :1 z :1) celles ci sont ensuite transmises au client

Serveur->Client : x:1,y:1,z:1.

Serveur->Client : Your turn

--- Le serveur passe la main au client, celui ci saisit ses coordonées transmises de la même manière, mais le serveur détermine si le coup est valide ou non, le client essaie de jouer aussi en 1 1 1 :

Client->Serveur : x:1,y:1,z:1.
Serveur->Client : Your turn

--- Le serveur lui repasse la main jusqu'à ce que le coup soit valide, il envoie alors un accusé au client

Client->Serveur : x:1,y:1,z:0.
Serveur->Client : Ok at x:1,y:1,z:0.

[…]

--- Le serveur aligne 3 pions	--- Le client aligne 3 pions
Serveur->Client : EOG:I won	Client->Serveur : EOG:You won

--- Le client interrompt la partie en fermant son socket

--- Le serveur détecte une erreur (recv retourne un buffer de taille nulle) et ferme à son tour son socket

XII. Bugs et observations

Les situations suivantes pourraient encore mener à des comportements instables :

- déconnexion du réseau de l'une des deux machines en cours de partie :
  nous l'avons rapidement testé et normalement la partie se ferme d'elle-même

- Si l'on ouvre à nouveau une partie alors que le programme vient de s'arrêter à cause d'une erreur réseau ou bien d'un appui sur Ctrl C, le socket du serveur ne peut pas être attaché immédiatement, nous avons mis en place une petite boucle d'attente pour contourner cet éventuel problème lié à la réallocation d'un socket par le système d'exploitation

- choix de couleurs identiques :
  nous ne retirons pas de la liste du choix de couleur la couleur utilisée par le premier joueur, si les deux joueurs choisissent la même couleur (en mode réseau particulièrement) le jeu est injouable

- En mode joueur contre ordinateur, avec tirage au sort du joueur commençant à placer ses pions, il semblerait que dans la version OpenGL l'ordinateur commence systématiquement…

- Lors de nos tests l'OpenGL avec mode alpha (transparence des pions) ne fonctionne pas correctement sous Linux.

- Le réseau reste à implémenter pour windows (à travers l'interface winsock)

XIII. Annexe : listing du code

Nous avons utilisé Doxygen pour générer une documentation du code mais les copier-collers du code à partir d'internet explorer transforment les '*' en '+'.

1) Global.h

```
00001 #ifndef GLOBAL_H_
00002 #define GLOBAL_H_
00003
00004 /************************************
00005  *            SYSTEME D'EXPLOITATION        *
00006  ************************************/
00007 #ifdef LINUX
00008 #define NETWORK 1
00009 #else
00010 #define WITHALPHA 1
00011 #endif
00012
00013 /************************
00014  *            INCLUSIONS            *
00015  ************************/
00016 #include <errno.h>
00017 #include <fcntl.h>
00018 #ifdef LINUX
00019 #include <unistd.h>
00020 #include <ctype.h> /* tolower() */
00021 #else
00022 #include <io.h>
00023 #endif
00024 #include <math.h>
00025 #include <stdio.h>
00026 #include <stdlib.h>
00027 #include <string.h>
00028 #include <sys/types.h>
00029 #include <sys/stat.h>
00030 #include <time.h>
00031 #include "opengl.h"
00032
00033 #ifdef WITHNETWORK
00034 #include "network.h"
00035 #endif
00036 /************************
00037  *            MACROS                *
00038  ************************/
00039 #define ContenuTab(tab, x, y, z)       *(tab + (z * NBX * NBY) + (y * NBX) + x)
00040 #ifdef LINUX
00041 #define EraseScreen()                    system("clear");
00042 #else
00043 #define EraseScreen()                    system("cls");
00044 #endif
00045
00046 /************************
00047  *            DEFINITIONS           *
00048  ************************/
00049 /* Définition des dimensions du plateau */
```

```
00050 #define NBX                         3
00051 #define NBY                         3
00052 #define NBZ                         3
00053 /* Caractères utilisés pour l'affichage des pions */
00054 #define PASDEPION                   0
00055 #define CASESELECT                  6
00056 /* Définition des handles correspondants aux I/O traditionnels */
00057 #define STDIN                       0
00058 #define STDOUT                      1
00059 #define STDERROR                    2
00060 /* Nombre de pions à aligner pou gagner */
00061 #define NBPIONSALIGN                3
00062 /* Booleen: */
00063 #define TRUE                        1
00064 #define FALSE                       0
00065 /* Définition des menus */
00066 #ifdef WITHOPENGL
00067 #ifndef WITHNETWORK
00068 #define MENUOFFSET 1
00069 #endif
00070 #endif
00071
00072 #ifdef WITHOPENGL
00073 #ifdef WITHNETWORK
00074 #define MENUOFFSET 2
00075 #endif
00076 #else
00077 #ifdef WITHNETWORK
00078 #define MENUOFFSET 1
00079 #else
00080 #define MENUOFFSET 0
00081 #endif
00082 #endif
00083
00084 #define LENMENUMAX                  40
00085 #define TITREMENUMAIN               "Menu principal:"
00086 #define DEUXJOUEURSUNPC             0
00087 #define MENU0                       "Deux joueurs sur le meme PC"
00088 #define UNJOUEUR                    1
00089 #define MENU1                       "Joueur contre ordinateur"
00090 #define DEUXJOUEURSRESEAU           2
00091 #define MENU2                       "Jeu en reseau"
00092 #define CHARGERPARTIE               1+MENUOFFSET
00093 #define MENU3                       "Charger une partie enregistree"
00094 #define QUITTER                     2+MENUOFFSET
00095 #define MENU4                       "Quitter le jeu"
00096 /*_____________________________________________________________*/
00097 #define TITREMENUIA                 "Sous-Menu IA: Choix de l'ordre des joueurs"
00098 #define JOUEURFIRST                 0
00099 #define MENUIA0                     "Le joueur joue en premier (easy)"
00100 #define IAFIRST                     1
00101 #define MENUIA1                     "L'IA joue en premier (very hard)"
00102 #define EGAL                        2
00103 #define MENUIA2                     "Tirage au sort de celui qui commencera"
00104 #define RETOUR                      3
00105 #define MENUIA3                     "Revenir au menu principal"
00106 /*_____________________________________________________________*/
00107 #define TITREMENURESEAU             "Sous-Menu Reseau:"
00108 #define SERVEUR                     0
00109 #define MENURESEAU0                 "Creer un serveur"
```

```
00110 #define CLIENT                  1
00111 #define MENURESEAU1                "Etre un client (se connecter au serveur)"
00112 #define RETOURRESEAU          2
00113 #define MENURESEAU2                "Revenir au menu principal"
00114 /*_____________________________________________________________________*/
00115 #define TITREMENUCOULEUR       "Choix de la couleur du pion:"
00116 #define ROUGE                  0
00117 #define COULEUR0                  "Rouge"
00118 #define VERT                   1
00119 #define COULEUR1                  "Vert"
00120 #define BLEU                   2
00121 #define COULEUR2                  "Bleu"
00122 #define JAUNE                  3
00123 #define COULEUR3                  "Jaune"
00124 #define BLANC                  4
00125 #define COULEUR4                  "Blanc"
00126 /* Définition de constantes diverses */
00127 #define SIZEBUFFER                256
00128 #define ENTETE                 "FICHIER DE SAUVEGARDE DU MORPION 3D"
00129 #define UNIMPLEMENTED          "Non implemente pour l'instant"
00130 #define NAMEIA                 "Artificial intelligence"
00131 #define IANUMBER                  3
00132 #define EXTENSIONSAUVEGARDE    ".sav"
00133
00134 /* Alias pour simplifier la compréhension */
00135 #define NO_SOCKET      NULL
00136
00137 /************************
00138  *          TYPES
00139  ************************/
00140 /* Définition des types propres au programme */
00141 typedef char* tplat;
00142
00143 /************************
00144  *          STRUCTURES
00145  ************************/
00146 /* Structure définissant les coordonnées d'une case du jeu */
00147 typedef struct COORD
00148 {
00149       unsigned char x, y, z;
00150 } COORD;
00151
00152 /* Définition des couleurs */
00153 typedef struct t_color
00154 {
00155       float red, green, blue;
00156 }t_color;
00157
00158 /* Structure définissant un joueur */
00159 typedef struct t_joueur
00160 {
00161       char* nom;
00162       int id;
00163       char pion;
00164       t_color couleur;
00165       unsigned char nbcoup;
00166 } t_joueur;
00167
00168 /* Structure des menus */
00169 typedef struct t_menu
```

```
00170 {
00171         char *titre;
00172         char *tabmenu[LENMENUMAX];
00173         int lenmenu;
00174 } t_menu;
00175
00176 /************************
00177  *              CONSTANTES              *
00178  ************************/
00179 /* Les différents menus */
00180 #ifdef WITHOPENGL
00181 #ifndef WITHNETWORK
00182 static const t_menu MAINMENU = {TITREMENUMAIN, {MENU0, MENU1, MENU3, MENU4}, 4};
00183 #endif
00184 #endif
00185
00186 #ifdef WITHOPENGL
00187 #ifdef WITHNETWORK
00188 static const t_menu MAINMENU = {TITREMENUMAIN, {MENU0, MENU1, MENU2, MENU3, MENU4}, 5};
00189 #endif
00190 #else
00191 #ifdef WITHNETWORK
00192 static const t_menu MAINMENU = {TITREMENUMAIN, {MENU0, MENU1, MENU2, MENU4}, 4};
00193 #else
00194 static const t_menu MAINMENU = {TITREMENUMAIN, {MENU0, MENU1, MENU4}, 3};
00195 #endif
00196 #endif
00197
00198 static const t_menu IAMENU = {TITREMENUIA, {MENUIA0, MENUIA1, MENUIA2, MENUIA3}, 4};
00199 #ifdef WITHNETWORK
00200 static const t_menu NETWORKMENU = {TITREMENURESEAU, {MENURESEAU0, MENURESEAU1, MENURESEAU2}, 3};
00201 #endif
00202 static const t_menu COULEURMENU = {TITREMENUCOULEUR, {COULEUR0, COULEUR1, COULEUR2, COULEUR3,
COULEUR4}, 5};
00203 /* Differents coordonnes remarquable */
00204 static const COORD MIDDLE = {NBX / 2, NBY / 2, NBZ / 2};
00205 static const COORD NOPOS = {NBX+1, NBY+1, NBZ+1};
00206 /*Couleurs de base => Ces couleurs sont exprimees en RGB, rapporte à l'inertvalle [0.0; 1.0] */
00207 static const t_color RED = {1.0f, 0.0f, 0.0f};
00208 static const t_color GREEN = {0.0f, 1.0f, 0.0f};
00209 static const t_color BLUE = {0.0f, 0.0f, 1.0f};
00210 static const t_color YELLOW = {0.9f, 1.0f, 0.0f};
00211 static const t_color WHITE = {1.0f, 1.0f, 1.0f};
00212 static const t_color NOPION = {0.5f, 0.4f, 0.4f};
00213 static const t_color FOND = {0.3f, 0.5f, 1.0f};
00214 static const t_color TEXTE = {0.5f, 0.0f, 0.6f};
00215 static const t_color IACOLOR = {1.0f, 0.5f, 0.3f};
00216 static const t_color SELECT = {0.2f, 0.3f, 0.4f};
00217
00218 /****************************************
00219  *              PROTOTYPES DES FONCTIONS:               *
00220  ****************************************/
00221 /* coord.c: */
00222 COORD ReadCoord(void);
00223 int SetCoord(char* tab, COORD place, t_joueur* joueur);
00224 void RemoveCoord(char* tab, COORD place);
00225 char IsNoPion(char* tab, COORD place);
00226 COORD FindFirstPion(char* tab, char pion);
00227 COORD CoordNear(char* tab, COORD pion);
00228 COORD SelectBestPlace(char* tab);
00229 /* file.c: */
```

```
00230 int CreateFile(char* nom);
00231 int OpenFile(char* nom);
00232 int PrintFile(void);
00233 int WriteDataPlayer(int handle, t_joueur* joueur);
00234 int SaveGame(tplat tab, t_joueur *joueur1, t_joueur *joueur2);
00235 int LoadGame(tplat tab, t_joueur *joueur1, t_joueur *joueur2);
00236 /* ia.h: */
00237 void MenuIA(tplat plateau, t_joueur *joueurA, t_joueur *joueurB, int *argc, char **argv);
00238 COORD FindCoordIA(char* tab, t_joueur* IA, t_joueur* adverse);
00239 /* init.c: */
00240 void InitMat(tplat tab);
00241 void InitPlayer(char id, t_joueur *joueur);
00242 void InitIA(t_joueur *joueur);
00243 /* jeu.c: */
00244 void InitGlGame(tplat tab, t_joueur *joueurA, t_joueur *joueurB, int* socket, int *argc, char **argv);
00245 void Game(tplat plateau, COORD select, t_joueur *joueur1, t_joueur *joueur2);
00246 void TextGame(tplat plateau, t_joueur* joueur1, t_joueur* joueur2, int* socket);
00247 /* main.c: */
00248 int main(int argc, char **argv);
00249 inline void restart();
00250 /* menu.c: */
00251 int DisplayMenu(t_menu menu, char ERASE);
00252 unsigned char ChooseMenu(int lenmenu);
00253 /* misc.c: */
00254 int WriteChar(int handle, char car);
00255 int WriteString(int handle, char* buf, int ret);
00256 char GetChar(int handle);
00257 char* GetString(int handle, int taille);
00258 char* CompleteExtension(char* nom, char* extension);
00259 char* CopyTab(char* destination, const char* source);
00260 void Verify(void);
00261 /* random.c: */
00262 void InitRandom(void);
00263 int Randomize(int inf, int sup);
00264 /* screen.c: */
00265 void Display(tplat tab, t_joueur* joueur1, t_joueur* joueur2);
00266 /* winner.c: */
00267 unsigned char IsWinner(char* tab, t_joueur* joueur, COORD lastplay);
00268 COORD FindPosForWin(char* tab, t_joueur* joueur, int* nb);
00269 COORD FindTwoPosWin(char* tab, t_joueur* joueur);
00270 #endif
```

2) Opengl.h

```
00001 #ifndef OPENGL_H_
00002 #define OPENGL_H_
00003
00004 /************************
00005  *            INCLUSIONS            *
00006  ************************/
00007 #ifndef LINUX
00008 #include <GL\glut.h>
00009 #else
00010 #include <GL/glut.h>
00011 #endif
00012 #include <math.h>
00013 #include <stdlib.h>
00014
00015 /************************
```

```
00016 *           DEFINITIONS           *
00017 ************************/
00018 /* Taille et position de la fenetre */
00019 #define POSX                0
00020 #define POSY                0
00021 #define WIDTH               800
00022 #define HEIGHT              600
00023 #define ECARTFENETRE    4
00024 #define MODE_DISPLAY    GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH
00025 #define WINDOWTITLE           "Morpion 3D"
00026 #define HEIGHTPOLICE    18
00027 /* Taille et position de la sous-fenetre 1 */
00028 #define POSXSUB1            ECARTFENETRE
00029 #define POSYSUB1            ECARTFENETRE
00030 #define WIDTHSUB1             w1 - 2 * ECARTFENETRE
00031 #define HEIGHTSUB1            ((h1 - ECARTFENETRE) / 2) - ECARTFENETRE
00032 /* Taille et position de la sous-fenetre 2 */
00033 #define POSXSUB2            POSXSUB1
00034 #define POSYSUB2            ((h1 - ECARTFENETRE) / 2) + ECARTFENETRE / 2
00035 #define WIDTHSUB2             w1 - 2 * ECARTFENETRE
00036 #define HEIGHTSUB2            HEIGHTSUB1 + ECARTFENETRE / 2
00037 /* Informations relatives à la sous-fenetre 2 */
00038 #define ECARTTAB              w1 / 40
00039 #define RATIO               0.8
00040 #define XCASE2              (WIDTHSUB2 - 3 * ECARTTAB) / 9
00041 #define YCASE2              ((HEIGHTSUB2 / 2.0) - (2.0 * ECARTTAB) - HEIGHTPOLICE) / 3.0
00042 #define XTAB                XCASE2 * NBX
00043 #define YTAB                YCASE2 * NBY
00044 #define ZCASE               1
00045 #define POSTEXTEX2            (k + 1) * ECARTTAB + k * XTAB + XTAB / 2.0 - glutBitmapLength((void*) font, (unsigned
char*)buffer) / 2
00046 #define POSTEXTEY2            YTAB + ECARTTAB + HEIGHTPOLICE
00047 /* Informations relatives à la sous-fenetre 1 */
00048 #define ECARTBORD             0.4
00049 #define RATIO1              0.6
00050 #define ALPHAVALUE            0.5
00051 #define POSTEXTEXWIN    (WIDTHSUB1 - 2 * ECARTBORD - glutBitmapLength((void*) font, (unsigned char*)buffer))
00052 #define POSTEXTEYWIN    (HEIGHTSUB1 - 2 * HEIGHTPOLICE)
00053 /* FIXME/ 2*/
00054
00055 /************************
00056 *           DEFINITIONS           *
00057 ************************/
00058 /* displayglobal_opengl.c: */
00059 void ChangeSize(int w,int h);
00060 void ChangeSize2(int w,int h);
00061 void RenderSceneAll(void);
00062 void RenderScene(void);
00063 /* displayS1_opengl.c: */
00064 void RenderScenesw1(void);
00065 void DrawCube(int i, int j, int k, GLenum state, char ARRETE);
00066 /* displayS2_opengl.c: */
00067 void RenderScenesw2(void);
00068 /* font_opengl.c: */
```

```
00069 void setOrthographicProjection(void);
00070 void resetPerspectiveProjection(void);
00071 void RenderBitmapString(float x, float y, void *font,char *string);
00072 /* keyboard_opengl.c: */
00073 void processSpecialKeys(int touche, int x, int y);
00074 void ProcessNormalKeys(unsigned char key, int x, int y);
00075 /* mouse_opengl.c: */
00076 void Mouse1(int button, int state,int x,int y);
00077 void MouseMotion1(int x,int y);
00078 void Mouse2(int button, int state, int x, int y);
00079 /* opengl_init.c: */
00080 void InitLibGlut(int* argc, char** argv);
00081 void InitWindow(void);
00082 void DestroyWindow(void);
00083
00084 #endif
```

3) Network.h

```
00001 #ifdef WITHNETWORK
00002 #include "global.h"
00003 #endif
00004 /* Sinon l'on compile les programmes de test des fonctions réseau */
00005 #include <sys/socket.h>
00006 #include <netinet/in.h>
00007
00008 #include <arpa/inet.h> /* inet_ntoa inet_pton */
00009
00010 /* Définitions : */
00011 #define MAXNETRETRY 10 /* Nombre maximum de tentative d'attachement du socket */
00012
00013 /*      Serveur.c              */
00014 char RunServer(int* newfd);
00015 void StopServer(int *newfd);
00016
00017 /*      Client.c               */
00018 char InitClient(int* fd);
00019 char Connect(int* fd, char* RemoteIP);
00020 void StopClient(int* fd);
00021
00022 /*      Network.c      */
00023 char NetSend(int* newfd, char* buffer, unsigned int bufsize);
00024 char NetReceive(int* newfd, char* buffer, unsigned int bufsize);
```

4) Main.c

```
00001 #include "global.h"
00002
00003 t_joueur joueur1, joueur2;
00004 char plat[NBX][NBY][NBZ];
00005 tplat  plateau=(tplat)plat;
00006 char ISSELECT, WINNER;
00007 char QUIT;
00008
00009 int main(int argc, char **argv)
00010 {
00011       QUIT = FALSE;
00012       joueur1.nom = NULL;
00013       joueur2.nom = NULL;
00014       atexit(restart);
00015
00016       Verify(); /* Vérifie quelques paramètres de compilation */
00017       do
00018       {
00019             /* Quel type de jeu veut-on effectuer? */
00020             ISSELECT = FALSE;
00021             WINNER = FALSE;
00022             switch (ChooseMenu(DisplayMenu(MAINMENU, TRUE)))
00023             {
00024             case DEUXJOUEURSUNPC:
00025                   InitMat(plateau);
00026                   InitPlayer(1, &joueur1);
00027                   InitPlayer(2, &joueur2);
00028 #ifdef WITHOPENGL
00029                   InitGlGame(plateau, &joueur1, &joueur2, NO_SOCKET, &argc, argv);
00030 #else
00031                   TextGame(plateau, &joueur1, &joueur2, NO_SOCKET);
00032 #endif
00033                   break;
00034             case UNJOUEUR:
00035                   InitMat(plateau);
00036                   if ((NBX != 3) || (NBY != 3) || (NBZ != 3))
00037                   {
00038                         WriteString(STDOUT, UNIMPLEMENTED, 1);
00039                         GetString(STDIN, SIZEBUFFER);
00040                   }
00041                   else
00042                         MenuIA(plateau, &joueur1, &joueur2, &argc, argv);
00043                   break;
00044 #ifdef WITHNETWORK
00045             case DEUXJOUEURSRESEAU:
00046                   {
00047                         int fdsocket;
00048                         char* key;
00049                         WriteString(STDOUT, "Version réseau",1);
00050                         puts("\ts: pour jouer en serveur\n\tc: pour jouer en client");
00051                         do {
00052                               key=GetString(STDIN,2);
00053                         }while(*key!='s'&&*key!='c');
```

```
00054
00055                   if(*key=='s')
00056                   {
00057                       char buffer[100];
00058                       InitPlayer(1,&joueur1);
00059                       if(!RunServer(&fdsocket))
00060                           break;
00061                       if(!NetReceive(&fdsocket, buffer, 100)) break;
00062                       joueur2.nom=(char*) malloc(SIZEBUFFER);
00063                       sscanf(buffer,"Hello, I am %s (%c).",joueur2.nom,&joueur2.pion);
00064                       joueur2.id=2;
00065                       joueur2.nbcoup=0;
00066                       printf("Le joueur %s jouant avec le pion %c s'est connecté\n",joueur2.nom, joueur2.pion);
00067                       sprintf(buffer,"Hello, I am %s (%c).",joueur1.nom,joueur1.pion);
00068                       NetSend(&fdsocket, buffer, strlen(buffer));
00069                       InitMat(plateau);
00070                       TextGame(plateau, &joueur1, &joueur2, &fdsocket);
00071                       free(joueur2.nom);
00072                       StopServer(&fdsocket);
00073                   }
00074                   else
00075                   {
00076                       char* ipserver;
00077                       char quit=0;
00078                       InitPlayer(2,&joueur1);
00079                       if(!InitClient(&fdsocket))
00080                           break;
00081
00082                       do{
00083                           WriteString(STDOUT, "Veuillez saisir l'adresse IP du serveur : (127.0.0.1)",1);
00084                           ipserver=GetString(STDIN,17);
00085                           if(*ipserver=='\0')
00086                           {
00087                               ipserver=(char*) malloc(17*sizeof(char));
00088                               strcpy(ipserver,"127.0.0.1");
00089                           }
00090                           if(*ipserver=='q' || Connect(&fdsocket,ipserver))
00091                               quit=1;
00092
00093                       }while(!quit);
00094                       InitMat(plateau);
00095                       joueur2.id=1;
00096                       {
00097                           char buffer[100];
00098                           char r;
00099                           sprintf(buffer,"Hello, I am %s (%c).",joueur1.nom,joueur1.pion);
00100                           NetSend(&fdsocket, buffer, strlen(buffer));
00101                           if(!NetReceive(&fdsocket, buffer, 100)) break;
00102                           joueur2.nom=(char*) malloc(SIZEBUFFER);
00103                           joueur2.nbcoup=0;
00104                           sscanf(buffer,"Hello, I am %s (%c).",joueur2.nom,&joueur2.pion);
00105                           printf("%s %c\n",joueur2.nom,joueur2.pion);
00106                           InitMat(plateau);
00107                           Display(plateau, &joueur1, &joueur2);
```

```
00108                           do {
00109                               COORD coords;
00110                               r=NetReceive(&fdsocket,buffer, 100);
00111                               if(*buffer=='x')
00112                               {
00113                                   sscanf(buffer,"x:%u,y:%u,z:%u.",
00114                                           &coords.x, &coords.y, &coords.z);
00115                   /* Nécessaire car la lecture doit se faire en entier sinon on obtient un caractère et non une valeur comprise entre 0 et 2 */
00116
00117                                   SetCoord(plateau,coords,&joueur2);
00118                                   EraseScreen();
00119                                   Display(plateau, &joueur1, &joueur2);
00120                                   joueur2.nbcoup++;
00121                               }
00122                               if(strncmp(buffer,"Your turn",10)==0)
00123                               {
00124
00125                                   char buf[20];
00126                                   coords=ReadCoord();
00127                                   sprintf(buf, "x:%u,y:%u,z:%u.",
00128                                         coords.x, coords.y, coords.z);
00129                                   NetSend(&fdsocket, buf,20);
00130                               }
00131                               if(strncmp(buffer,"OK at",5)==0)
00132                               {
00133                                   sscanf(buffer,"OK at x:%u,y:%u,z:%u.",
00134                                           &coords.x,&coords.y,&coords.z);
00135                                   SetCoord(plateau,coords,&joueur1);
00136                                   EraseScreen();
00137                                   Display(plateau, &joueur1, &joueur2);
00138                                   joueur1.nbcoup++;
00139                               }
00140                               if(strncmp(buffer,"EOG:I won",9)==0)
00141                               {
00142                                   printf("Vous avez perdu! Après %d tour%c\n",joueur1.nbcoup,(joueur1.nbcoup>1)?'s':' ');
00143                                   WriteString(STDOUT, "Appuyez sur ENTREE pour continuer...", 1);
00144                                   GetString(STDIN, SIZEBUFFER);
00145                               }
00146                               if(strncmp(buffer,"EOG:You won",11)==0)
00147                               {
00148                                   printf("Vous avez gagné! Après %d tour%c\n",joueur1.nbcoup,(joueur1.nbcoup>1)?'s':' ');
00149                                   WriteString(STDOUT, "Appuyez sur ENTREE pour continuer...", 1);
00150                                   GetString(STDIN, SIZEBUFFER);
00151                               }
00152                           } while(strncmp(buffer,"EOG",3)!=0&&r);
00153
00154                       }
00155                       StopClient(&fdsocket);
00156                       free(ipserver);
00157                   }
00158                   free(key);
00159
00160           }
00161
```

```
00162                   break;
00163 #endif
00164 #ifdef WITHOPENGL
00165           case CHARGERPARTIE:
00166                   if (LoadGame(plateau, &joueur1, &joueur2))
00167                           InitGlGame(plateau, &joueur1, &joueur2, NO_SOCKET, &argc, argv);
00168                   break;
00169 #endif
00170           case QUITTER:
00171                   QUIT = TRUE;
00172                   break;
00173           default:
00174                   break;
00175           }
00176           if(joueur1.nom!=NULL)
00177           {
00178                   free(joueur1.nom);
00179                   free(joueur2.nom);
00180                   joueur1.nom = NULL;
00181                   joueur2.nom = NULL;
00182           }
00183       } while (QUIT == FALSE);
00184       exit(0);
00185 }
00186
00187 void restart()
00188 {
00189       if(!QUIT)
00190           main(0, NULL);
00191       /* Relance le menu principal lorsque la fermeture de la fenêtre opengl a complètement fermé le programme */
00192 }
```

5) Misc.c

```
00001 #include "global.h"
00002
00003 /*****************************************************
00004  *WriteChar:                                         *
00005  *     Ecrit un caractère donné sur la sortie spécifié *
00006  *****************************************************/
00007 int WriteChar(int handle, char car)
00008 {
00009       if (!write(handle, &car, sizeof(char)))
00010           return 0;
00011       return 1;
00012 }
00013
00014 /*************************************************************
00015  *WriteString:                                                *
00016  *     Ecrit une chaine de caractère donné sur la sortie spécifiée    *
00017  *     Met ret \n derriere la chaine                                  *
00018  *     /!\:S'arrete au premier'\n' rencontré dans la chaine           *
00019  *************************************************************/
00020 int WriteString(int handle, char* buf, int ret)
```

```
00021 {
00022       int i = 0;
00023
00024       while (*(buf + i) != '\n' && *(buf + i) != '\0')
00025       {
00026             if (!WriteChar(handle, *(buf + i)))
00027                   return FALSE;
00028             i++;
00029       }
00030       for (i = 0; i < ret; i++)
00031       {
00032             if (!WriteChar(handle, '\n'))
00033                   return 0;
00034       }
00035       return TRUE;
00036 }
00037
00038 /*****************************************************************
00039  *GetChar:                                                       *
00040  *     Récupere un caractère sur la sortie spécifiée et le renvoie     *
00041  *****************************************************************/
00042 char GetChar(int handle)
00043 {
00044       char c;
00045
00046       if(!read(handle, &c, sizeof(char)))
00047             return -1;
00048       return c;
00049 }
00050
00051 /*****************************************************************
00052  *GetString:                                                     *
00053  *     Recupere une chaine de caractère de longueur maximale taille    *
00054  *****************************************************************/
00055 char* GetString(int handle, int taille)
00056 {
00057       char* str, *p;
00058
00059       str = (char*) malloc(taille * sizeof(char));
00060       p = str;
00061       do
00062       {
00063             *p = GetChar(handle);
00064             p++;
00065       } while (*(p - 1) != '\n' && *(p - 1) != '\0' && (p - str) < taille);
00066       *(p - 1) = '\0';
00067       return strdup(str);
00068 }
00069
00070 /*****************************************************************************************
00071  *ComleteExtension:                                                                      *
00072  *     Ajoute au om du fichier son extension, au cas ou l'utilisateur ne l'aurait pas mise     *
00073  *****************************************************************************************/
00074 char* CompleteExtension(char* nom, char* extension)
```

```
00075 {
00076       char* pos;
00077
00078       pos = strstr(nom, ".");
00079       if (pos != NULL)
00080             *pos = '\0';
00081       strcat(nom, extension);
00082       return strdup(nom);
00083 }
00084
00085 /*******************************************
00086  *CopyTab:                                 *
00087  *     Copie un tableau dans un autre tableau  *
00088  *******************************************/
00089 char* CopyTab(char* destination, const char* source)
00090 {
00091       unsigned char i, j, k;
00092
00093       for (k = 0; k < NBZ; k++)
00094             for (j = 0; j < NBY; j++)
00095                   for (i = 0; i < NBX; i++)
00096                         ContenuTab(destination, i, j, k) = ContenuTab(source, i, j, k);
00097       return destination;
00098 }
00099
00100 /*****************************************************
00101  *Verify:                                            *
00102  *     Verifie si les valeurs du header sont plausibles       *
00103  *****************************************************/
00104 void Verify(void)
00105 {
00106       char TEST = FALSE;
00107
00108       if (NBX < 1 || NBY < 1 || NBZ < 1)
00109       {
00110             TEST = TRUE;
00111             WriteString(STDOUT, "Plateau trop petit", 1);
00112             GetString(STDIN, SIZEBUFFER);
00113       }
00114       if (NBPIONSALIGN < 2)
00115       {
00116             TEST = TRUE;
00117             WriteString(STDOUT, "Alignement mal configure", 1);
00118             GetString(STDIN, SIZEBUFFER);
00119       }
00120       if (TEST)
00121             exit(1);
00122 }
```

6) Init.c

```
00001 #include "global.h"
00002
00003 /*************************************************
```

```
00004 *InitMat:                                                    *
00005 *     Initialise le plateau de jeu à 3 dimensions            *
00006 *     en le remplissant de 0                                 *
00007 ****************************************************/
00008 void InitMat(tplat tab)
00009 {
00010       char i, j, k;
00011
00012       for (k = 0; k < NBZ; k++)
00013             for (j = 0; j < NBY; j++)
00014                   for (i = 0; i < NBX; i++)
00015                         ContenuTab(tab, i, j, k) = PASDEPION;
00016       return;
00017 }
00018
00019 /************************************************
00020 *InitPlayer:                                                 *
00021 *     Initialise les donnees concernant un joueur     *
00022 ************************************************/
00023 void InitPlayer(char id, t_joueur *joueur)
00024 {
00025       char couleur;
00026       EraseScreen();
00027       WriteString(STDOUT, "Initialisation d'un joueur:", 1);
00028       printf("Nom du joueur: [joueur%d]\n", id);
00029       joueur->nom = GetString(STDIN, SIZEBUFFER);
00030       if(*(joueur->nom)=='\0')
00031       {
00032
00033             char tbuf[2];
00034             joueur->nom = (char*) malloc(8*sizeof(char));
00035             strcpy(joueur->nom, "joueur");
00036             tbuf[0]=id+'0'; /* conversion entier >-1, <10 en ascii */
00037             tbuf[1]='\0';
00038             strcat(joueur->nom,tbuf);
00039       }
00040       couleur=ChooseMenu(DisplayMenu(COULEURMENU, FALSE));
00041       switch(couleur)
00042       {
00043       case ROUGE:
00044             joueur->couleur = RED;
00045             joueur->pion = 'r';
00046             break;
00047       case BLEU:
00048             joueur->couleur = BLUE;
00049             joueur->pion = 'b';
00050             break;
00051       case VERT:
00052             joueur->couleur = GREEN;
00053             joueur->pion = 'g';
00054             break;
00055       case JAUNE:
00056             joueur->couleur = YELLOW;
00057             joueur->pion = 'y';
```

```
00058               break;
00059       case BLANC:
00060               joueur->couleur = WHITE;
00061               joueur->pion = 'w';
00062               break;
00063       default:
00064               break;
00065       }
00066       joueur->id = id;
00067       joueur->nbcoup = 0;
00068       return;
00069 }
00070
00071 /****************************
00072  *InitIA:                   *
00073  *     Initialisation de l'IA  *
00074  ****************************/
00075 void InitIA(t_joueur *joueur)
00076 {
00077       joueur->nbcoup = 0;
00078       joueur->nom = strdup(NAMEIA);
00079       joueur->pion = 'I';
00080       joueur->couleur = IACOLOR;
00081       joueur->id = IANUMBER;
00082       return;
00083 }
```

7) Jeu.c

```
00001 #include "global.h"
00002
00003 t_joueur *joueur;
00004 extern char WINNER;
00005
00006 /******************************************
00007  *InitGlGame:                             *
00008  *     Initialise le jeu et la fenetre OpenGL  *
00009  ******************************************/
00010 #ifdef WITHOPENGL
00011 void InitGlGame(tplat plateau, t_joueur *joueur1, t_joueur *joueur2, int* socket, int *argc, char **argv)
00012 {
00013       joueur = joueur1;
00014
00015       InitLibGlut(argc, argv);
00016       InitWindow();
00017       glutShowWindow();
00018       glutMainLoop();
00019 }
00020 #endif
00021 /*********************************************************************
00022  *Game:                                                              *
00023  *     Fonction de jeu, place un pion et vérifie si on a un vainqueur  *
00024  *     Pour placer son pion, 2 possibilités:                          *
00025  *           la souris (fonction Mouse2)                              *
00026  *           le changement d'image si c'est à l'IA de jouer           *
```

```
00027 ***************************************************************************/
00028 void Game(tplat plateau, COORD select, t_joueur *joueur1, t_joueur *joueur2)
00029 {
00030       /* On met le pion sur la case spécifiée */
00031       ContenuTab(plateau, select.x, select.y, select.z) = joueur->id;
00032       /* Incrémentation du nombre de coup joué */
00033       joueur->nbcoup++;
00034       /* On recherche si on a un vainqueur */
00035       if (IsWinner(plateau, joueur, select))
00036             WINNER = TRUE;
00037       else
00038             joueur = (joueur == joueur1) ? joueur2 : joueur1;
00039       return;
00040 }
00041
00042 /*****************************************************************************
00043  * TextGame:
00044  *      Version texte et réseau du coeur du jeu, détermine qui joue
00045  *       fait vérifier les coordonnées, gère éventuellement le réseau et l'IA
00046  *****************************************************************************/
00047 void TextGame(tplat plateau, t_joueur* joueur1, t_joueur* joueur2, int* socket)
00048 {
00049       COORD place;
00050       char EOG=FALSE;
00051       joueur = joueur2;
00052       /* Nécessaire d'initialiser à une valeur, influe sur le joueur jouant le premier dans le mode contre l'ordinateur */
00053       do
00054       {
00055             joueur = (joueur == joueur1)? joueur2 : joueur1;
00056             EraseScreen();
00057             WriteString(STDOUT, "Morpion 3D realise par Savon et Seb", 1);
00058 #ifdef WITHOPENGL
00059             WriteString(STDOUT, "Pour quitter la partie tapez la lettre \'Q\', pour sauvegarder la partie tapez la lettre \'S\'", 2);
00060 #endif
00061             Display(plateau, joueur1, joueur2);
00062 #ifdef WITHNETWORK
00063             if(socket!=NO_SOCKET && joueur==joueur2)
00064             {
00065                   /* Au tour du joueur distant */
00066                   COORD coords;
00067                   char netbuf[100];
00068
00069                   do {
00070                         NetSend(socket,"Your turn", 10);
00071                         if(!NetReceive(socket,netbuf,100)) return;
00072                         sscanf(netbuf,"x:%d,y:%d,z:%d.",
00073                                     &coords.x,&coords.y,&coords.z);
00074                   }while(SetCoord(plateau, coords, joueur) != TRUE);
00075                   sprintf(netbuf,"OK at x:%d,y:%d,z:%d.",coords.x,coords.y,coords.z);
00076                   NetSend(socket,netbuf,strlen(netbuf));
00077                   joueur2->nbcoup++;
00078                   printf("%s a joué en %d %d %d\n",
00079                               joueur2->nom,coords.x,coords.y,coords.z);
00080                   EOG = IsWinner(plateau, joueur, coords);
```

```
00081                   if(EOG)
00082                   {
00083                           sleep(1);
00084                           NetSend(socket,"EOG:You won",12);
00085                           joueur=joueur2;
00086                           break;
00087                   }
00088
00089                   continue;
00090           }
00091 #endif
00092           do
00093           {
00094                   WriteString(STDOUT, joueur->nom, 1);
00095
00096           if(joueur->id!=IANUMBER)
00097                   place = ReadCoord();
00098           else
00099                   place = FindCoordIA(plateau, joueur, ((joueur == joueur1) ? joueur2 : joueur1));
00100           } while (SetCoord(plateau, place, joueur) != TRUE);
00101 #ifdef WITHNETWORK
00102           if(socket!=NO_SOCKET && joueur==joueur1)
00103           {
00104                   char netbuf[20];
00105                   sprintf(netbuf, "x:%u,y:%u,z:%u.",
00106                           place.x, place.y, place.z);
00107                   NetSend(socket, netbuf,20);
00108           }
00109 #endif
00110           /* On incrémente le nombre de coup joue */
00111           joueur->nbcoup++;
00112           /* On regarde si le joueur a gagne */
00113           EOG = IsWinner(plateau, joueur, place);
00114 #ifdef WITHNETWORK
00115           if(socket!=NO_SOCKET && joueur==joueur1 && EOG)
00116           {
00117                   sleep(1);
00118                   NetSend(socket,"EOG:I won",12);
00119           }
00120 #endif
00121       } while((!EOG) && (joueur1->nbcoup + joueur2->nbcoup) < (NBX * NBY * NBZ));
00122       EraseScreen();
00123       WriteString(STDOUT, "Morpion 3D realise par Savon et Seb", 1);
00124       WriteString(STDOUT, "Pour quitter la partie tapez la lettre \'Q\', pour sauvegarder la partie tapez la lettre \'S\'", 2);
00125       Display(plateau, joueur1, joueur2);
00126       if (joueur1->nbcoup + joueur2->nbcoup >=(NBX*NBY*NBZ))
00127           puts("Egalite parfaite, aucun joueur n'a gagne apres cette partie très serree");
00128
00129       else
00130           printf("%s a gagne en %d coups\n", joueur->nom, joueur->nbcoup);
00131       WriteString(STDOUT, "Appuyez sur ENTREE pour continuer...", 1);
00132       GetString(STDIN, SIZEBUFFER);
00133       return;
00134 }
```

8) Coord.c

```
00001 #include "global.h"
00002
00003 /*****************************************************
00004  *ReadCoord:                                          *
00005  *     Lit les coordonnées d'une case et les renvoie   *
00006  *     Ces coordonnées ne peuvent etre >=NB ou < 0      *
00007  *     Cette fonction servira pour du debuggage         *
00008  *     Pas en mode graphique avec OpenGL                *
00009  *****************************************************/
00010 COORD ReadCoord(void)
00011 {
00012       COORD place;
00013       /* on initialise à une valeur entière non signée impossible dans le jeu */
00014       place.x = NBX+1;
00015       place.y = NBY+1;
00016       place.z = NBZ+1;
00017       WriteString(STDOUT, "Saisissez les coordonnees de l'endroit ou vous voulez jouer:\n", 1);
00018       while (place.x >= NBX)
00019       {
00020             WriteString(STDOUT, "X = ", 0);
00021             place.x = GetChar(STDIN);
00022             place.x -= '0';
00023             /* Pour supprimer le retour a la ligne */
00024             GetString(STDIN, SIZEBUFFER);
00025       }
00026       while (place.y >= NBY)
00027       {
00028             WriteString(STDOUT, "Y = ", 0);
00029             place.y = GetChar(STDIN);
00030             place.y -= '0';
00031             /* Pour supprimer le retour a la ligne */
00032             GetString(STDIN, SIZEBUFFER);
00033       }
00034       while (place.z >= NBZ)
00035       {
00036             WriteString(STDOUT, "Z = ", 0);
00037             place.z = GetChar(STDIN);
00038             place.z -= '0';
00039             /* Pour supprimer le retour a la ligne */
00040             GetString(STDIN, SIZEBUFFER);
00041       }
00042       return place;
00043 }
00044
00045 /*******************************************************
00046  *SetCoord:                                             *
00047  *     Place le pion du joueur aux coordonées spécifiées      *
00048  *******************************************************/
00049 int SetCoord(char* tab, COORD place, t_joueur* joueur)
00050 {
00051       if (IsNoPion(tab, place))
00052       {
00053             ContenuTab(tab, place.x, place.y, place.z) = joueur->id;
```

```
00054               return TRUE;
00055         }
00056         return FALSE;
00057 }
00058
00059 /*********************************
00060  *RemoveCoord:                          *
00061  *      Retire un pion d'une case       *
00062  *********************************/
00063 void RemoveCoord(char* tab, COORD place)
00064 {
00065         ContenuTab(tab, place.x, place.y, place.z) = PASDEPION;
00066         return;
00067 }
00068
00069 /*************************************************************************
00070  *IsNoPion:                                                                *
00071  *      Renvoie 1 si il n'y a pas de pion sur la case passee en parametre       *
00072  *************************************************************************/
00073 char IsNoPion(char* tab, COORD place)
00074 {
00075         if (ContenuTab(tab, place.x, place.y, place.z) == PASDEPION)
00076               return TRUE;
00077         return FALSE;
00078 }
00079
00080 /*******************************************************
00081  *FindFisrtPos:                                              *
00082  *      Renvoie la postion du premier pion sur le plateau       *
00083  *******************************************************/
00084 COORD FindFirstPion(char* tab, char pion)
00085 {
00086         unsigned char i, j, k;
00087         COORD place;
00088
00089         for (k = 0; k < NBZ; k++)
00090               for (j = 0; j < NBY; j++)
00091                     for (i = 0; i < NBX; i++)
00092                     {
00093                           if (ContenuTab(tab, i, j, k) == pion)
00094                           {
00095                                 place.x = i;
00096                                 place.y = j;
00097                                 place.z = k;
00098                                 return place;
00099                           }
00100                     }
00101         return NOPOS;
00102 }
00103
00104 /*****************************************************************************
00105  *CoordNear:                                                                  *
00106  *      Renvoie les coordonnees d'une case entourant la case donnee en parametre et ce  *
00107  *      de façon aleatoire. Il faut que cette case touche pleinement la case donnee           *
```

```
00108 ********************************************************************************/
00109 COORD CoordNear(char* tab, COORD pion)
00110 {
00111       /* Il ne peut y avoir que 6 cases adjacentes a une case donnée */
00112       COORD place[6], temp;
00113       char i;
00114       unsigned char cmpt;
00115
00116       cmpt = 0;
00117       temp = pion;
00118       for (i = -1; i <= 1; i++)
00119       {
00120             if (i == 0)
00121                   continue;
00122             temp.x = (char) (pion.x + i);
00123             /* Il faut verifier que l'on peut bien mettre un pion sur une case (pas occupée et dans le tableau) */
00124             if (IsNoPion(tab, temp) && temp.x < NBX)
00125                   place[cmpt++] = temp;
00126       }
00127       temp = pion;
00128       for (i = -1; i <= 1; i++)
00129       {
00130             if (i == 0)
00131                   continue;
00132             temp.y = (char) (pion.y + i);
00133             if (IsNoPion(tab, temp) && temp.y < NBY)
00134                   place[cmpt++] = temp;
00135       }
00136       temp = pion;
00137       for (i = -1; i <= 1; i++)
00138       {
00139             if (i == 0)
00140                   continue;
00141             temp.z = (char) (pion.z + i);
00142             if (IsNoPion(tab, temp) && temp.z < NBZ)
00143                   place[cmpt++] = temp;
00144       }
00145       return place[Randomize(0, (int) cmpt - 1)];
00146 }
00147
00148
00149 /*******************************************************
00150  *  SelectBestPlace:                                   *
00151  *   Renvoie les coordonnées d'un des meilleurs        *
00152  *   emplacements à jouer pour l'IA                    *
00153  *    ************************************************/
00154 COORD SelectBestPlace(char *tab)
00155 {
00156       COORD *place;
00157       unsigned char i, j;
00158
00159 /* Ce sont ces tableaux qui limitent l'IA à un jeu par 3*3*3 cases!
00160       Ce tableau de coordonees contient les cases qui ont 6 possibilites
00161       d'alignement si la case MIDDLE est prise */
```

```
00162       COORD tab6[8] = {
00163                                   {0, 0, 0}, {0, 0, (NBZ-1)}, {0, (NBY-1), 0}, {0, (NBY-1), (NBZ-1)},
00164                                   {(NBX-1), 0, 0}, {(NBX-1), (NBY-1), 0}, {(NBX-1), 0, (NBZ-1)}, {(NBX-1), (NBY-1), (NBZ-1)}
00165                               };
00166       /* Ce tableau de coordonees contient les cases qui ont 4 possibilites
00167       d'alignement si la case MIDDLE est prise */
00168       COORD tab4[4] = {
00169                                   {0, (NBY-1)/2, (NBZ-1)/2}, {(NBX-1), (NBY-1)/2, (NBZ-1)/2},
00170                                   {(NBX-1)/2, 0, (NBZ-1)/2}, {(NBX-1)/2, (NBY-1), (NBZ-1)/2}
00171                               };
00172       /* Ce tableau de coordonees contient les cases qui ont 3 possibilites
00173       d'alignement si la case MIDDLE est prise */
00174       COORD tab3[14] = {
00175                                   {0, (NBY-1)/2, 0}, {(NBX-1), (NBY-1)/2, 0}, {(NBX-1)/2, 0, 0}, {(NBX-1)/2, (NBY-1), 0},
00176                                   {0, (NBY-1)/2, (NBZ-1)}, {(NBX-1), (NBY-1)/2, (NBZ-1)}, {(NBX-1)/2, 0, (NBZ-1)}, {(NBX-1)/2, (NBY-
1), (NBZ-1)},
00177                                   {0, 0, (NBZ-1)/2}, {0, (NBY-1), (NBZ-1)/2}, {(NBX-1), 0, (NBY-1)/2}, {(NBX-1), (NBY-1), (NBZ-1)/2},
00178                                   {(NBX-1)/2, (NBY-1)/2, 0}, {(NBX-1)/2, (NBY-1)/2, (NBZ-1)}
00179                               };
00180
00181       place = (COORD*) malloc(8 * sizeof(char));
00182       j = 0;
00183       for (i = 0; i < 8; i++)
00184       {
00185           if (IsNoPion(tab, tab6[i]))
00186           {
00187               place[j] = tab6[i];
00188               j++;
00189           }
00190       }
00191       if (j)
00192           return place[Randomize(0, j - 1)];
00193
00194       place = (COORD*) realloc(place, 4 * sizeof(char));
00195       j = 0;
00196       for (i = 0; i < 4; i++)
00197       {
00198           if (IsNoPion(tab, tab4[i]))
00199           {
00200               place[j] = tab4[i];
00201               j++;
00202           }
00203       }
00204       if (j)
00205           return place[Randomize(0, j - 1)];
00206
00207       place = (COORD*) realloc(place, 14 * sizeof(char));
00208       j = 0;
00209       for (i = 0; i < 14; i++)
00210       {
00211           if (IsNoPion(tab, tab3[i]))
00212           {
00213               place[j] = tab3[i];
00214               j++;
```

```
00215             }
00216       }
00217       if (j)
00218             return place[Randomize(0, j - 1)];
00219
00220       /* Impossible, ou le plateau est completement rempli!!! */
00221       return NOPOS;
00222 }
```

9) Menu.c

```
00001 #include "global.h"
00002
00003 /*******************************************************
00004  *DisplayMenu:                                          *
00005  *     Affiche un menu et renvoie la longueur de celui-ci     *
00006  *******************************************************/
00007 int DisplayMenu(t_menu menu, char ERASE)
00008 {
00009       unsigned char i;
00010       char buffer[SIZEBUFFER];
00011
00012       if (ERASE)
00013             EraseScreen();
00014       WriteString(STDOUT, menu.titre, 1);
00015       for (i = 0; i < menu.lenmenu; i++)
00016       {
00017             sprintf(buffer, "\t%i: %s", i, menu.tabmenu[i]);
00018             WriteString(STDOUT, buffer, 1);
00019       }
00020       WriteChar(STDOUT, '\n');
00021       return i;
00022 }
00023
00024 /*************************************************************
00025  *ChooseMenu:                                                  *
00026  *     Retourne le numéro du menu choisit par l'utilisateur    *
00027  *************************************************************/
00028 unsigned char ChooseMenu(int lenmenu)
00029 {
00030       unsigned char c;
00031
00032       do
00033       {
00034             WriteString(STDOUT, "Quel menu voulez-vous choisir?\t", 0);
00035             c = (unsigned char) (GetChar(STDIN) - '0');
00036             /* Pour supprimer le retour à la ligne */
00037             GetString(STDIN, SIZEBUFFER);
00038             if (c < lenmenu)
00039                   break;
00040             WriteString(STDOUT, "Menu choisi invalide", 1);
00041       } while (c >= lenmenu);
00042       return c;
00043 }
```

10) Random.c

```
00001 #include "global.h"
00002
00003 /****************************************
00004  *InitRandom:                           *
00005  *     Initialisation de la fonction rand     *
00006  ****************************************/
00007 void InitRandom(void)
00008 {
00009       time_t date;
00010
00011       time(&date);
00012       srand(date);
00013 }
00014
00015 /************************************************
00016  *Randomize:                                     *
00017  *     Tire un nombre aleatoire entre inf et sup      *
00018  ************************************************/
00019 int Randomize(int inf, int sup)
00020 {
00021       int x;
00022
00023       InitRandom();
00024       if (inf <= sup)
00025             return inf;
00026       x = rand() % (sup - inf + 1) + inf;
00027       return x;
00028 }
```

11) Screen.c

```
00001 #include "global.h"
00002
00003 /****************************************
00004  *Display:                              *
00005  *     Affiche à l'écran le tableau de jeu     *
00006  ****************************************/
00007 void Display(tplat tab, t_joueur* joueur1, t_joueur* joueur2)
00008 {
00009       unsigned char i, j, k;
00010       char c;
00011
00012       for (k = 0; k < NBZ; k++)
00013       {
00014             for (j = 0; j < NBY; j++)
00015             {
00016                   for (i = 0; i < NBX; i++)
00017                   {      if(ContenuTab(tab,i,j,k)==joueur1->id)
00018                               c = joueur1->pion;
00019                         else
00020                         if (ContenuTab(tab,i,j,k)==joueur2->id)
00021                               c = joueur2->pion;
00022                         else
00023                               c = '.';
00024                         WriteChar(STDOUT,c);
```

```
00025                   }
00026                   WriteChar(STDOUT, '\n');
00027
00028               }
00029               WriteChar(STDOUT, '\n');
00030       }
00031       return;
00032 }
```

12) Winner.c

```
00001 #include "global.h"
00002
00003 /*****************************************************************************************************
00004  *IsWinner:                                                                                          *
00005  *      Cherche si un joueur a gagné.                                                                 *
00006  *      Pour que ce joueur est gagné, il faut que le dernier pion qu'il est posé soit dans l'alignement *
00007  *      Renvoie cmpt qui vaut 0 si aucun alignement est trouve                                        *
00008  *REM:  Fonction longue car on tient compte du fait que l'on peut faire varier la taille du plateau    *
00009  *            et le nombre de pions à aligner                                                         *
00010  *REM:  Si NBPIONALIGN=3 et NBX=NBY=NBZ=3 (morpion classique)                                         *
00011  *            il y a 3 façons d'aligner ses pions     (X:pion que l'on vient de jouer; x:pion deja present    *
00012  *                  ..Xxx ou .xXx. ou xxX.. donc 5 cases à tester (i varie de -2 a 2, donc 5 cas different) *
00013  *****************************************************************************************************/
00014 unsigned char IsWinner(tplat tab, t_joueur* joueur, COORD lastplay)
00015 {
00016       char i, nb, nb2;
00017       unsigned char cmpt;
00018
00019       /* cmpt est utilise pour regarder le nombre de fois que l'on gagne en jouant sur la case lastplay */
00020       cmpt = 0;
00021       /* On doit trouver deux pions autour de celui que l'on vient de poser et qui soient alignés */
00022       nb = 0;
00023       /* Test sur les x: */
00024       for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00025       {
00026             if ((lastplay.x + i) < 0 || (lastplay.x + i) >= NBX)
00027                   continue;
00028             /* On va passer par le pion que l'on vient de mettre quand i = 0 */
00029             if (ContenuTab(tab, (lastplay.x + i), lastplay.y, lastplay.z) == joueur->id)
00030                   nb++;
00031             else
00032                   nb = 0;
00033             if (nb >= NBPIONSALIGN)
00034             {
00035                   cmpt++;
00036                   break;
00037             }
00038       }
00039
00040       /* Test sur les y: */
00041       nb = 0;
00042       for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00043       {
00044             if ((lastplay.y + i) < 0 || (lastplay.y + i) >= NBY)
```

```
00045                 continue;
00046           /* On va passer par le pion que l'on vient de mettre */
00047           if (ContenuTab(tab, lastplay.x, ((lastplay.y + i)), lastplay.z) == joueur->id)
00048                 nb++;
00049           else
00050                 nb = 0;
00051           if (nb >= NBPIONSALIGN)
00052           {
00053                 cmpt++;
00054                 break;
00055           }
00056     }
00057
00058     /* Test sur les z: */
00059     nb = 0;
00060     for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00061     {
00062           if ((lastplay.z + i) < 0 || (lastplay.z + i) >= NBZ)
00063                 continue;
00064           /* On va passer par le pion que l'on vient de mettre */
00065           if (ContenuTab(tab, lastplay.x, lastplay.y, (lastplay.z + i)) == joueur->id)
00066                 nb++;
00067           else
00068                 nb = 0;
00069           if (nb >= NBPIONSALIGN)
00070           {
00071                 cmpt++;
00072                 break;
00073           }
00074     }
00075
00076 /* Test sur les diagonales (2 diagonales par plan au plus) et il y a 3 plans xy; xz et yz
00077      Plan xy: */
00078     nb = 0;
00079     nb2 = 0;
00080     for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00081     {
00082           if ((lastplay.x + i) < 0 || (lastplay.x + i) >= NBX)
00083                 continue;
00084           if ((lastplay.y + i) >= 0 && (lastplay.y + i) < NBY)
00085           {
00086                 if (ContenuTab(tab, (lastplay.x + i), (lastplay.y + i), lastplay.z) == joueur->id)
00087                       nb++;
00088                 else
00089                       nb = 0;
00090           }
00091           if ((lastplay.y - i) >= 0 && (lastplay.y - i) < NBY)
00092           {
00093                 if (ContenuTab(tab, (lastplay.x + i), (lastplay.y - i), lastplay.z) == joueur->id)
00094                       nb2++;
00095                 else
00096                       nb2 = 0;
00097           }
00098           if (nb >= NBPIONSALIGN)
```

```
00099           {
00100                   cmpt++;
00101                   break;
00102           }
00103           if (nb2 >= NBPIONSALIGN)
00104           {
00105                   cmpt++;
00106                   break;
00107           }
00108   }
00109
00110   /* Plan xz: */
00111   nb = 0;
00112   nb2 = 0;
00113   for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00114   {
00115           if ((lastplay.x + i) < 0 || (lastplay.x + i) >= NBX)
00116                   continue;
00117           if ((lastplay.z + i) >= 0 && (lastplay.z + i) < NBY)
00118           {
00119                   if (ContenuTab(tab, (lastplay.x + i), lastplay.y, (lastplay.z + i)) == joueur->id)
00120                           nb++;
00121                   else
00122                           nb = 0;
00123           }
00124           if ((lastplay.z - i) >= 0 && (lastplay.z - i) < NBY)
00125           {
00126                   if (ContenuTab(tab, (lastplay.x + i), lastplay.y, (lastplay.z - i)) == joueur->id)
00127                           nb2++;
00128                   else
00129                           nb2 = 0;
00130           }
00131           if (nb >= NBPIONSALIGN)
00132           {
00133                   cmpt++;
00134                   break;
00135           }
00136           if (nb2 >= NBPIONSALIGN)
00137           {
00138                   cmpt++;
00139                   break;
00140           }
00141   }
00142
00143   /* Plan yz: */
00144   nb = 0;
00145   nb2 = 0;
00146   for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00147   {
00148           if ((lastplay.y + i) < 0 || (lastplay.y + i) >= NBX)
00149                   continue;
00150           if ((lastplay.z + i) >= 0 && (lastplay.z + i) < NBZ)
00151           {
00152                   if (ContenuTab(tab, lastplay.x, (lastplay.y + i), (lastplay.z + i)) == joueur->id)
```

```
00153                   nb++;
00154               else
00155                   nb = 0;
00156           }
00157           if ((lastplay.z - i) >= 0 && (lastplay.z - i) < NBZ)
00158           {
00159               if (ContenuTab(tab, lastplay.x, (lastplay.y + i), (lastplay.z - i)) == joueur->id)
00160                   nb2++;
00161               else
00162                   nb2 = 0;
00163           }
00164           if (nb >= NBPIONSALIGN)
00165           {
00166               cmpt++;
00167               break;
00168           }
00169           if (nb2 >= NBPIONSALIGN)
00170           {
00171               cmpt++;
00172               break;
00173           }
00174       }
00175
00176       /*Si on vient de jouer le pion au mileu du plateau (case 1 1 1) on a 4 diagonales possibles supplémentaires
00177       Ou si on vient de jouer un pion sur les 8 coins du cube (alors une seule diagonale possible
00178       On aurait pu tester case par case puisqu'il n'y avait que 4 possibilité, mais on perdait la possibilité
00179       de changer le nombre de pions à aligner et les dimension du "plateau" de jeu */
00180       nb = 0;
00181       nb2 = 0;
00182       for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00183       {
00184           if ((lastplay.x + i) < 0 || (lastplay.x + i) >= NBX || (lastplay.y + i) < 0 || (lastplay.y + i) >= NBY)
00185               continue;
00186           if ((lastplay.z + i) >= 0 && (lastplay.z + i) < NBZ)
00187           {
00188               if (ContenuTab(tab, (lastplay.x + i), (lastplay.y + i), (lastplay.z + i)) == joueur->id)
00189                   nb++;
00190               else
00191                   nb = 0;
00192           }
00193           if ((lastplay.z - i) >= 0 && (lastplay.z - i) < NBZ)
00194           {
00195               if (ContenuTab(tab, (lastplay.x + i), (lastplay.y + i), (lastplay.z - i)) == joueur->id)
00196                   nb2++;
00197               else
00198                   nb2 = 0;
00199           }
00200           if (nb >= NBPIONSALIGN)
00201           {
00202               cmpt++;
00203               break;
00204           }
00205           if (nb2 >= NBPIONSALIGN)
00206           {
```

```
00207                   cmpt++;
00208                   break;
00209               }
00210       }
00211
00212       nb = 0;
00213       nb2 = 0;
00214       for (i = -(NBPIONSALIGN - 1); i <= NBPIONSALIGN - 1; i++)
00215       {
00216           if ((lastplay.x + i) < 0 || (lastplay.x + i) >= NBX || (lastplay.y - i) < 0 || (lastplay.y - i) >= NBY)
00217               continue;
00218           if ((lastplay.z + i) >= 0 && (lastplay.z + i) < NBZ)
00219           {
00220               if (ContenuTab(tab, ((lastplay.x + i)), ((lastplay.y - i)), ((lastplay.z + i))) == joueur->id)
00221                   nb++;
00222               else
00223                   nb = 0;
00224           }
00225           if ((lastplay.z - i) >= 0 && (lastplay.z - i) < NBZ)
00226           {
00227               if (ContenuTab(tab, ((lastplay.x + i)), ((lastplay.y - i)), ((lastplay.z - i))) == joueur->id)
00228                   nb2++;
00229               else
00230                   nb2 = 0;
00231           }
00232           if (nb >= NBPIONSALIGN)
00233           {
00234               cmpt++;
00235               break;
00236           }
00237           if (nb2 >= NBPIONSALIGN)
00238           {
00239               cmpt++;
00240               break;
00241           }
00242       }
00243
00244       /* Si on arrive ici, il n'y a pas d'alignement de NBPIONSALIGN */
00245       return cmpt;
00246 }
00247
00248 /************************************************
00249  *FindPosForWin:                                *
00250  *     Trouve si il y a une possibilite de gagner     *
00251  *     nb est le nombre de fois ou l'on gagne         *
00252  *     Renvoi une position aleatoire ou l'on gagne    *
00253  ************************************************/
00254
00255 COORD FindPosForWin(tplat tab, t_joueur* joueur, int* nb)
00256 {
00257       COORD place, coordtemp[NBX * NBY * NBZ];
00258       unsigned char i, j, k;
00259       char temp[NBX][NBY][NBZ];
00260
```

```
00261       *nb = 0;
00262       CopyTab((tplat) temp, tab);
00263       for (k = 0; k < NBZ; k++)
00264       {
00265             place.z = k;
00266             for (j = 0; j < NBY; j++)
00267             {
00268                   place.y = j;
00269                   for (i = 0; i < NBX; i++)
00270                   {
00271                         place.x = i;
00272                         if (IsNoPion((tplat) temp, place))
00273                         {
00274                               SetCoord((tplat) temp, place, joueur);
00275                               if (IsWinner((tplat) temp, joueur, place))
00276                               {
00277                                     coordtemp[(*nb)] = place;
00278                                     (*nb)++;
00279                               }
00280                               RemoveCoord((tplat) temp, place);
00281                         }
00282                   }
00283             }
00284       }
00285       if (*nb)
00286       {
00287             i = Randomize(0, (*nb) - 1);
00288             return coordtemp[i];
00289       }
00290       else
00291             return NOPOS;
00292 }
00293
00294 /*****************************************************************************************
00295  *FindTwoPosWin:                                                                         *
00296  *      Cherche une case qui permet d'avoir la possibilite de gagner sur 2 cases differentes    *
00297  *****************************************************************************************/
00298 COORD FindTwoPosWin(tplat tab, t_joueur* joueur)
00299 {
00300       COORD place, *coordtemp;
00301       unsigned char i, j, k;
00302       int nb = 0;
00303       char* temp, cmpt;
00304
00305       cmpt = -1;
00306       coordtemp = (COORD*) malloc(NBX * NBY * NBZ * sizeof(COORD));
00307       temp = (char*) malloc(NBX * NBY * NBZ * sizeof(char));
00308       temp = CopyTab(temp, tab);
00309       for (k = 0; k < NBZ; k++)
00310       {
00311             place.z = k;
00312             for (j = 0; j < NBY; j++)
00313             {
00314                   place.y = j;
```

```
00315                   for (i = 0; i < NBX; i++)
00316                   {
00317                           place.x = i;
00318                           if (IsNoPion(temp, place))
00319                           {
00320                                   SetCoord(temp, place, joueur);
00321                                   FindPosForWin(temp, joueur, &nb);
00322                                   RemoveCoord(temp, place);
00323                                   if (nb >= 2)
00324                                   {
00325                                           cmpt++;
00326                                           *(coordtemp + cmpt) = place;
00327                                   }
00328                           }
00329                   }
00330           }
00331   }
00332   if (cmpt != -1)
00333           return coordtemp[Randomize(0, cmpt)];
00334   return NOPOS;
00335 }
```

13) File.c

```
00001 #include "global.h"
00002
00003 /*****************************************************************************************
00004  *CreateFile:                                                                            *
00005  *      Cree un fichier vide ou efface le fichier existant (apres accord de l'utilisateur)      *
00006  *****************************************************************************************/
00007 int CreateFile(char* nom)
00008 {
00009   int handle;
00010   char c;
00011
00012   WriteString(STDOUT, "Sous quel nom voulez-vous sauvegardez votre fichier?", 1);
00013   nom = GetString(STDIN, SIZEBUFFER);
00014   CompleteExtension(nom, EXTENSIONSAUVEGARDE);
00015 #ifdef LINUX
00016   if ((handle = open(nom, O_EXCL | O_CREAT | O_WRONLY, S_IWUSR | S_IRUSR)) == -1)
00017 #else
00018   if ((handle = open(nom, _O_EXCL | _O_CREAT | _O_WRONLY | _O_TEXT, _S_IWRITE | _S_IREAD)) == -1)
00019 #endif
00020   {
00021           /* Si le fichier existe déjà */
00022           if (errno == EEXIST)
00023           {
00024                   WriteString(STDOUT, "Fichier deja existant, voulez-vous l'ecraser? (o/n)", 1);
00025                   do
00026                   {
00027                           c = GetChar(STDIN);
00028                           GetString(STDIN, SIZEBUFFER);
00029                           c = (char) tolower(c);
00030                   } while (c != 'o' && c != 'n');
00031                   if (c == 'o')
```

```
00032 #ifdef LINUX
00033                         handle = open(nom, O_TRUNC | O_RDWR);
00034                         /* Un seul mode d'accès sous Linux */
00035 #else
00036                         handle = open(nom, _O_TRUNC | _O_RDWR | _O_TEXT);
00037 #endif
00038               }
00039       }
00040       return handle;
00041 }
00042
00043 /*****************************************************************************************************
00044  *OpenFile:                                                                                          *
00045  *     Ouvre un fichier sans en effacer le contenu (utile pour lire ce que conttient le fichier!!!)    *
00046  *     La donnee nom contiendra à la fin de la fonction le nom du fichier ouvert                       *
00047  *****************************************************************************************************/
00048 int OpenFile(char* nom)
00049 {
00050       char nbtentative = 0;
00051       int handle;
00052
00053       do
00054       {
00055             if(!PrintFile())
00056                   return -1;
00057             WriteString(STDOUT, "Quel fichier voulez-vous ouvrir?", 1);
00058             nom = GetString(STDIN, SIZEBUFFER);
00059             nom = CompleteExtension(nom, EXTENSIONSAUVEGARDE);
00060
00061 #ifdef LINUX
00062             handle = open(nom, O_RDWR);
00063 #else
00064             handle = open(nom, _O_RDWR | _O_TEXT);
00065 #endif
00066             nbtentative++;
00067       } while (handle == -1 && nbtentative >= 4);
00068       if (handle == -1)
00069       {
00070             WriteString(STDOUT, "Erreur d'ouverture du fichier", 1);
00071             GetString(STDIN, SIZEBUFFER);
00072       }
00073       return handle;
00074 }
00075
00076 /*****************************************************************************
00077  *PrintFile:                                                                 *
00078  *     Affiche tous les fichiers portant l'extension ".sav" dans le repertoire courant *
00079  *****************************************************************************/
00080
00081 int PrintFile(void)
00082 {
00083 #if LINUX == 1
00084       system("ls --format=single-column *.sav");
00085       return TRUE;
```

```
00086 #else
00087       char nom[SIZEBUFFER];
00088       struct _finddata_t sav_file;
00089     long hFile;
00090
00091       EraseScreen();
00092       if( (hFile = _findfirst( "*.sav", &sav_file )) == -1L )
00093       {
00094             WriteString(STDOUT, "Pas de fichier de sauvegarde dans le repertoire du jeu", 1);
00095             GetString(STDIN, SIZEBUFFER);
00096             return FALSE;
00097       }
00098       else
00099       {
00100             WriteString(STDOUT, "Fichier de sauvegarde present dans le repertoire:", 1);
00101             sprintf(nom, "%-12s\n", sav_file.name);
00102             WriteString(STDOUT, nom, 1);
00103             /* Trouve le reste des .dic du repertoire*/
00104             while( _findnext( hFile, &sav_file ) == 0 )
00105             {
00106                   sprintf(nom, "%-12s\n", sav_file.name);
00107                   WriteString(STDOUT, nom, 1);
00108             }
00109              _findclose(hFile);
00110       }
00111       return TRUE;
00112 #endif
00113 }
00114
00115 /*************************************************************************
00116  *WriteDataPlayer:                                                       *
00117  *     Ecrit les donnees d'un joueur sur la sortie representee par handle     *
00118  *************************************************************************/
00119 int WriteDataPlayer(int handle, t_joueur* joueur)
00120 {
00121       char buffer[SIZEBUFFER];
00122
00123       if (!WriteChar(handle, '\n'))
00124             return FALSE;
00125       sprintf(buffer, "%s", joueur->nom);
00126       if (!WriteString(handle, buffer, 1))
00127             return FALSE;
00128       if (!WriteChar(handle, joueur->id))
00129             return FALSE;
00130       WriteChar(handle, '\n');
00131       sprintf(buffer, "%c", joueur->pion);
00132       if (!WriteString(handle, buffer, 1))
00133             return FALSE;
00134       sprintf(buffer, "%i", joueur->nbcoup);
00135       if (!WriteString(handle, buffer, 1))
00136             return FALSE;
00137       sprintf(buffer, "%f:%f:%f", joueur->couleur.red, joueur->couleur.green, joueur->couleur.blue);
00138       if (!WriteString(handle, buffer, 0))
00139             return FALSE;
```

```
00140
00141       return TRUE;
00142 }
00143
00144 /*****************************************************************************
00145  *SaveGame:                                                                  *
00146  *     Sauvegarde une partie en ecrivant dans un fichier le "plateau" du jeu   *
00147  *     Et les caracteristiques des 2 joueurs (joueur qui jouait en premier)    *
00148  *****************************************************************************/
00149 int SaveGame(char* tab, t_joueur* joueur1, t_joueur* joueur2)
00150 {
00151       char* nom;
00152       int handle, i, j, k;
00153
00154       EraseScreen();
00155       if ((handle = CreateFile(nom)) == -1)
00156             return FALSE;
00157       WriteString(STDOUT, "Sauvegarde...", 1);
00158       /* Pour verifier qu'il s'agisse d'un fichier valide */
00159       if (!WriteString(handle, ENTETE, 1))
00160       {
00161             close(handle);
00162             unlink(nom);
00163             return FALSE;
00164       }
00165       for(k = 0; k < NBZ; k++)
00166             for (j = 0; j < NBY; j++)
00167                   for (i = 0; i < NBX; i++)
00168                         if (!WriteChar(handle, ContenuTab(tab, i, j, k)))
00169                         {
00170                               close(handle);
00171                               unlink(nom);
00172                               return FALSE;
00173                         }
00174       if (!WriteDataPlayer(handle, joueur1))
00175       {
00176             close(handle);
00177             unlink(nom);
00178             return FALSE;
00179       }
00180       if (!WriteDataPlayer(handle, joueur2))
00181       {
00182             close(handle);
00183             unlink(nom);
00184             return FALSE;
00185       }
00186       close(handle);
00187       WriteString(STDOUT, "Sauvergarde terminee", 1);
00188       return TRUE;
00189 }
00190
00191 /*****************************************************************
00192  *LoadGame:                                                      *
00193  *     Recupere une partie sauvegardee                           *
```

```
00194 *          +Verification du type de fichier                              *
00195 *          +Initialisation de la matrice avec les valeurs du fichier     *
00196 *          +Initialisation des joueurs                                   *
00197 *          +Renvoie le joueur qui doit jouer                             *
00198 ****************************************************************/
00199 int LoadGame(char* tab, t_joueur *joueur1, t_joueur *joueur2)
00200 {
00201       char* nom, *buffer;
00202       int i, j, k, handle;
00203
00204       if ((handle = OpenFile(nom)) == -1)
00205             return FALSE;
00206       buffer = GetString(handle, SIZEBUFFER);
00207       i = ((strlen(buffer) > strlen(ENTETE)) ? strlen(ENTETE) : strlen(buffer));
00208       if (strncmp(buffer, ENTETE, i))
00209       {
00210             WriteString(STDOUT, "Fichier incompatible", 1);
00211             GetString(STDIN, SIZEBUFFER);
00212             return FALSE;
00213       }
00214       /* On lit la matrice */
00215       for (k = 0; k < NBZ; k++)
00216             for (j = 0; j < NBY; j++)
00217                   for (i = 0; i < NBX; i++)
00218                         ContenuTab(tab, i, j, k) = GetChar(handle);
00219       /* On lit le retour à la ligne */
00220       GetChar(handle);
00221       /* On lit les infos concernant le joueur1 */
00222       joueur1->nom = (char*) malloc(SIZEBUFFER);
00223       joueur1->nom = GetString(handle, SIZEBUFFER);
00224       joueur1->id = GetChar(handle);
00225       GetChar(handle);
00226       buffer = GetString(handle, SIZEBUFFER);
00227       joueur1->pion =*buffer;
00228       buffer = GetString(handle, SIZEBUFFER);
00229       joueur1->nbcoup = (unsigned char) atoi(buffer);
00230       buffer = GetString(handle, SIZEBUFFER);
00231       sscanf(buffer, "%f:%f:%f",&joueur1->couleur.red,&joueur1->couleur.green,&joueur1->couleur.blue);
00232       /* On lit le retour à la ligne */
00233       joueur2->nom = (char*) malloc(SIZEBUFFER);
00234       joueur2->nom = GetString(handle, SIZEBUFFER);
00235       printf("a lu %s %s\n", joueur1->nom, joueur2->nom);
00236       joueur2->id = GetChar(handle);
00237       GetChar(handle);
00238       buffer = GetString(handle, SIZEBUFFER);
00239       joueur2->pion=*buffer;
00240       buffer = GetString(handle, SIZEBUFFER);
00241       joueur2->nbcoup = (unsigned char) atoi(buffer);
00242       buffer = GetString(handle, SIZEBUFFER);
00243       sscanf(buffer, "%f:%f:%f",&joueur2->couleur.red,&joueur2->couleur.green,&joueur2->couleur.blue);
00244       close(handle);
00245       return TRUE;
00246 }
```

14) Ia.c

```
00001 #include "global.h"
00002
00003 /*****************************************************************************************************
00004  *MenuIA:                                                                                            *
00005  *      Affiche un menu pour choisir si on compte jouer en premier ou si on laisse la main à l'IA      *
00006  *****************************************************************************************************/
00007 void MenuIA(tplat plateau, t_joueur *joueurA, t_joueur *joueurB, int *argc, char **argv)
00008 {
00009       int j;
00010
00011       switch (ChooseMenu(DisplayMenu(IAMENU, TRUE)))
00012       {
00013       case JOUEURFIRST:
00014             InitPlayer(1, joueurA);
00015             InitIA(joueurB);
00016
00017 #ifdef WITHOPENGL
00018             InitGlGame(plateau, joueurA, joueurB, NO_SOCKET, argc, argv);
00019 #else
00020             TextGame(plateau, joueurA, joueurB, NO_SOCKET);
00021 #endif
00022             break;
00023       case IAFIRST:
00024             InitPlayer(2, joueurB);
00025             InitIA(joueurA);
00026 #ifdef WITHOPENGL
00027             InitGlGame(plateau, joueurA, joueurB, NO_SOCKET, argc, argv);
00028 #else
00029             TextGame(plateau, joueurA, joueurB, NO_SOCKET);
00030 #endif
00031             break;
00032       case EGAL:
00033             j = Randomize(0, 1);
00034             j ? InitPlayer(1, joueurA) : InitIA(joueurA);
00035             j ? InitIA(joueurB) : InitPlayer(2, joueurB);
00036 #ifdef WITHOPENGL
00037             InitGlGame(plateau, joueurA, joueurB, NO_SOCKET, argc, argv);
00038 #else
00039             TextGame(plateau, joueurA, joueurB, NO_SOCKET);
00040 #endif
00041             break;
00042       case RETOUR:
00043             break;
00044       default:
00045             break;
00046       }
00047       return;
00048 }
00049
00050 /*************************************************************************
00051  *FindCoordIA:                                                           *
00052  *      coeur de l'IA, cette fonction détermine les priorités de jeu de l'IA    *
00053  *************************************************************************/
```

```
00054 COORD FindCoordIA(tplat tab, t_joueur* IA, t_joueur* adverse)
00055 {
00056       COORD place;
00057       int winner;
00058
00059       /* Si la case du milieu est vide (au 1er tour de l'IA, cette case est prise) */
00060       if (IsNoPion(tab, MIDDLE))
00061             return MIDDLE;
00062       /* Si l'IA a commence et que c'est son deuxieme tour */
00063       if (IA->nbcoup == 1 && adverse->nbcoup == 1)
00064       {
00065             place = FindFirstPion(tab, adverse->id);
00066             /* Normalement on ne rentre pas dans le test, l'adversaire a joue un et un seul coup
00067             Donc il a un pion sur le plateau que nous avons trouvé */
00068             if (place.x >= NBX || place.y >= NBY || place.z >= NBZ)
00069                   return NOPOS;
00070             /* On joue a cote du pion juste posé */
00071             place = CoordNear(tab, place);
00072             return place;
00073       }
00074       else
00075       {
00076             /* On regarde si on peut gagner */
00077             place = FindPosForWin(tab, IA, &winner);
00078             if (place.x < NBX && place.y < NBY && place.z < NBZ)
00079                   return place;
00080             /* l'adversaire peut gagner au tour suivant? */
00081             place = FindPosForWin(tab, adverse, &winner);
00082             if (place.x < NBX && place.y < NBY && place.z < NBZ)
00083                   return place;
00084             /* Je regarde si je peux aligner 2*2pions */
00085             place = FindTwoPosWin(tab, IA);
00086             if (place.x < NBX && place.y < NBY && place.z < NBZ)
00087                   return place;
00088             /* Je regarde si l'adversaire peut en mettant au tour suivant un pion
00089             avoir 2 possibilites de gagner dans 2 tours */
00090             place = FindTwoPosWin(tab, adverse);
00091             if (place.x < NBX && place.y < NBY && place.z < NBZ)
00092                   return place;
00093             /* Si aucun des cas ne sont possibles */
00094             return SelectBestPlace(tab);
00095       }
00096 }
```

15) Opengl_init.c

```
00001 #include "global.h"
00002
00003 int mainWindow, subWindow1, subWindow2;
00004 extern int w1, h1;
00005
00006 /*****************************************************
00007  *InitLibGlut:                                       *
00008  *     Initialisation de la librairie GLUT pour OpenGL *
00009  *****************************************************/
```

```
00010 void InitLibGlut(int* argc, char** argv)
00011 {
00012       glutInit(argc, argv);
00013       /* Buffer de profondeur, double buffer et fenetre en mode RGBA */
00014 #ifdef WITHALPHA
00015       glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA | GLUT_ALPHA);
00016 #else
00017       glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE);
00018 #endif
00019       return;
00020 }
00021
00022 /*************************************************************
00023  *InitWindow:                                                *
00024  *     Initialisation et creation de la fenetre du jeu en OpenGL      *
00025  *************************************************************/
00026 void InitWindow(void)
00027 {
00028       atexit(DestroyWindow);
00029       /* Information sur la fenetre (emplacement d'origine, taille et titre) */
00030       glutInitWindowPosition(POSX, POSY);
00031       w1=WIDTH;
00032       h1=HEIGHT;
00033
00034       glutInitWindowSize(WIDTH, HEIGHT);
00035       mainWindow = glutCreateWindow(WINDOWTITLE);
00036       glClearColor(FOND.red, FOND.green, FOND.blue, 1.0);
00037       /* Les fonctions a appeler si tel ou tel evenement arrive */
00038       glutReshapeFunc(ChangeSize);
00039       glutDisplayFunc(RenderScene);
00040       glutIdleFunc(RenderSceneAll);
00041       glutSpecialFunc(processSpecialKeys);
00042       glutKeyboardFunc(ProcessNormalKeys);
00043       /* On divise la fenetre en 3 parties
00044       Dans cette partie, on affichera un cube en 3D qui tournera */
00045       subWindow1 = glutCreateSubWindow(mainWindow, POSXSUB1, POSYSUB1, WIDTHSUB1, HEIGHTSUB1);
00046       /* On initialise le alpha blending pour jouer avec la transparence */
00047 #ifdef WITHALPHA
00048       glEnable(GL_BLEND);
00049 #endif
00050       /* Pour mettre de la couleur sur les surfaces dessinées */
00051       glEnable(GL_COLOR_MATERIAL);
00052       glutDisplayFunc(RenderScenesw1);
00053       glutSpecialFunc(processSpecialKeys);
00054       glutKeyboardFunc(ProcessNormalKeys);
00055       glutMouseFunc(Mouse1);
00056       glutMotionFunc(MouseMotion1);
00057       /* Dans cette 2nde partie, on affichera le cube sous forme de 3 coupes planes */
00058       subWindow2 = glutCreateSubWindow(mainWindow, POSXSUB2, POSYSUB2, WIDTHSUB2, HEIGHTSUB2);
00059       glutDisplayFunc(RenderScenesw2);
00060       glutSpecialFunc(processSpecialKeys);
00061       glutKeyboardFunc(ProcessNormalKeys);
00062       glutMouseFunc(Mouse2);
00063       return;
```

```
00064 }
00065
00066 void DestroyWindow(void)
00067 {
00068       if (mainWindow)
00069       {
00070             glutDestroyWindow(subWindow1);
00071             glutDestroyWindow(subWindow2);
00072             glutDestroyWindow(mainWindow);
00073       }
00074       return;
00075 }
00076
```

16) Displayglobal_opengl.c

```
00001 #include "global.h"
00002
00003 extern int mainWindow, subWindow1, subWindow2;
00004 int w1, h1;
00005
00006 /*****************************************************************************
00007  *ChangeSize:                                                                *
00008  *     Pour changer la taille des fenetres si la fenetre principale est redimensionnée *
00009  *****************************************************************************/
00010 void ChangeSize(int w,int h)
00011 {
00012       /* Pour ne pas diviser par 0 */
00013       if(h <= 0)
00014             h = 1;
00015       if (w <= 0)
00016             w = 1;
00017       w1 = w;
00018       h1 = h;
00019       /* On affiche les differentes fenetres avec leur nouvelle taille */
00020       glutSetWindow(subWindow1);
00021       glutPositionWindow(POSXSUB1, POSYSUB1);
00022       glutReshapeWindow(WIDTHSUB1, HEIGHTSUB1);
00023       ChangeSize2(WIDTHSUB1, HEIGHTSUB1);
00024       glutSetWindow(subWindow2);
00025       glutPositionWindow(POSXSUB2, POSYSUB2);
00026       glutReshapeWindow(WIDTHSUB2, HEIGHTSUB2);
00027       return;
00028 }
00029
00030 /*****************************************************************
00031  *ChangeSize2:                                                   *
00032  *     Change la vue de la fenetre pour s'adapter au nouveau ratio     *
00033  *     Valable uniquement pour la 3D car on adapte la perspective      *
00034  *****************************************************************/
00035 void ChangeSize2(int w, int h)
00036 {
00037       if (w < h)
00038             glViewport(0,(h-w)/2,w,w);
00039       else
```

```
00040               glViewport((w-h)/2,0,h,h);
00041
00042 }
00043
00044 /*******************************************************
00045  *RenderSceneAll:                                      *
00046  *      On réaffiche toutes les fenetres et sous fenetres      *
00047  *******************************************************/
00048 void RenderSceneAll(void)
00049 {
00050
00051       RenderScene();
00052       RenderScenesw1();
00053       RenderScenesw2();
00054       return;
00055 }
00056
00057 /*************************************
00058  *RenderScene:                        *
00059  *      Affichage de la fenetre principale     *
00060  *************************************/
00061 void RenderScene(void)
00062 {
00063       glutSetWindow(mainWindow);
00064       glClear(GL_COLOR_BUFFER_BIT);
00065       glutSwapBuffers();
00066       return;
00067 }
```

17) displayS1_opengl.c

```
00001 #include "global.h"
00002
00003 extern int subWindow1;
00004 extern int angley, anglex;
00005 extern int w1, h1;
00006 extern t_joueur joueur1, joueur2, *joueur;
00007 extern tplat plateau;
00008 extern char WINNER;
00009
00010 /*****************************************************
00011  *RenderScenesw1:                                     *
00012  *      Fonction d'affichage de la sous fenetre 1             *
00013  *      Affichage d'un cube composée de 27 autres cubes *
00014  *      qui pourra tourner avec la souris (clic gauche) *
00015  *****************************************************/
00016 void RenderScenesw1(void)
00017 {
00018       int i, j, k;
00019
00020       glutSetWindow(subWindow1);
00021       /* effacement de l'image avec la couleur de fond */
00022       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
00023       glPushMatrix();
00024       /* Si on a un vainqueur, on l'affiche à l'écran */
```

```
00025       if (WINNER)
00026       {
00027             int font;
00028             char buffer[SIZEBUFFER];
00029
00030             font = (int) GLUT_BITMAP_HELVETICA_18;
00031             glPopMatrix();
00032             glColor3f(TEXTE.red, TEXTE.green, TEXTE.blue);
00033             setOrthographicProjection();
00034             glPushMatrix();
00035             glLoadIdentity();
00036             sprintf(buffer, "%s a gagne en %i coups", joueur->nom, joueur->nbcoup);
00037             RenderBitmapString(POSTEXTEXWIN, POSTEXTEYWIN, (void *)font, buffer);
00038             glPopMatrix();
00039             resetPerspectiveProjection();
00040       }
00041       glLoadIdentity();
00042       glEnable(GL_DEPTH_TEST);
00043       glRotatef(-angley,1.0,0.0,0.0);
00044       glRotatef(anglex,0.0,-1.0,0.0);
00045       /* Pour le alpha blending, il faut faire attention à l'ordre dans lequel on dessine le cube */
00046       for (k = 0; k < NBZ; k++)
00047       {
00048             for (j = 0; j < NBY; j++)
00049             {
00050                   for (i = 0; i < NBX; i++)
00051                   {
00052                         if (ContenuTab((char*) plateau, i, j, k) == PASDEPION)
00053                               DrawCube(i, j, k, GL_LINE_LOOP, FALSE);
00054                         else
00055                         {
00056                               DrawCube(i, j, k, GL_QUADS, FALSE);
00057                               DrawCube(i, j, k, GL_LINE_LOOP, TRUE);
00058                         }
00059                   }
00060             }
00061       }
00062       glPopMatrix();
00063       glFlush();
00064       /* On echange les buffers */
00065       glutSwapBuffers();
00066       return;
00067 }
00068
00069 /*********************************
00070  *DrawCube:                              *
00071  *     Dessine les 6 faces du cube     *
00072  *********************************/
00073 void DrawCube(int i, int j, int k, GLenum state, char ARRETE)
00074 {
00075       float xcase, ycase, zcase;
00076
00077       xcase = (2.0 - 2.0 * ECARTBORD) / NBX;
00078       ycase = (2.0 - 2.0 * ECARTBORD) / NBY;
```

```
00079       zcase = (2.0 - 2.0 * ECARTBORD) / NBZ;
00080       /* face 1 du cube */
00081       glBegin(state);
00082 #ifdef WITHALPHA
00083           if (ARRETE)
00084               glColor4f(0.0f, 0.0f, 0.0f, ALPHAVALUE);
00085           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00086               glColor4f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue, ALPHAVALUE);
00087           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00088               glColor4f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue, ALPHAVALUE);
00089           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00090               glColor4f(SELECT.red, SELECT.green, SELECT.blue, ALPHAVALUE);
00091           else
00092               glColor4f(NOPION.red, NOPION.green, NOPION.blue, ALPHAVALUE);
00093 #else
00094           if (ARRETE)
00095               glColor3f(0.0f, 0.0f, 0.0f);
00096           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00097               glColor3f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue);
00098           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00099               glColor3f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue);
00100           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00101               glColor3f(SELECT.red, SELECT.green, SELECT.blue);
00102           else
00103               glColor3f(NOPION.red, NOPION.green, NOPION.blue);
00104
00105 #endif
00106
00107           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + k * zcase);
00108           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + k * zcase);
00109           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + k *
zcase);
00110           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + k * zcase);
00111       glEnd();
00112       /* face 2 du cube */
00113       glBegin(state);
00114 #ifdef WITHALPHA
00115           if (ARRETE)
00116               glColor4f(0.0f, 0.0f, 0.0f, ALPHAVALUE);
00117           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00118               glColor4f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue, ALPHAVALUE);
00119           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00120               glColor4f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue, ALPHAVALUE);
00121           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00122               glColor4f(SELECT.red, SELECT.green, SELECT.blue, ALPHAVALUE);
00123           else
00124               glColor4f(NOPION.red, NOPION.green, NOPION.blue, ALPHAVALUE);
00125 #else
00126           if (ARRETE)
00127               glColor3f(0.0f, 0.0f, 0.0f);
00128           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00129               glColor3f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue);
00130           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00131               glColor3f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue);
```

```
00132           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00133                   glColor3f(SELECT.red, SELECT.green, SELECT.blue);
00134           else
00135                   glColor3f(NOPION.red, NOPION.green, NOPION.blue);
00136 #endif
00137           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + k * zcase);
00138           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + k * zcase);
00139           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + (k + RATIO1) *
zcase);
00140           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + (k + RATIO1) * zcase);
00141       glEnd();
00142       /* face 3 du cube */
00143       glBegin(state);
00144 #ifdef WITHALPHA
00145           if (ARRETE)
00146                   glColor4f(0.0f, 0.0f, 0.0f, ALPHAVALUE);
00147           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00148                   glColor4f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue, ALPHAVALUE);
00149           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00150                   glColor4f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue, ALPHAVALUE);
00151           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00152                   glColor4f(SELECT.red, SELECT.green, SELECT.blue, ALPHAVALUE);
00153           else
00154                   glColor4f(NOPION.red, NOPION.green, NOPION.blue, ALPHAVALUE);
00155 #else
00156           if (ARRETE)
00157                   glColor3f(0.0f, 0.0f, 0.0f);
00158           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00159                   glColor3f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue);
00160           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00161                   glColor3f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue);
00162           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00163                   glColor3f(SELECT.red, SELECT.green, SELECT.blue);
00164           else
00165                   glColor3f(NOPION.red, NOPION.green, NOPION.blue);
00166 #endif
00167           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + (k + RATIO1) * zcase);
00168           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + (k + RATIO1) *
zcase);
00169           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + (k +
RATIO1) * zcase);
00170           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + (k + RATIO1) *
zcase);
00171       glEnd();
00172       /* face 4 du cube */
00173       glBegin(state);
00174 #ifdef WITHALPHA
00175           if (ARRETE)
00176                   glColor4f(0.0f, 0.0f, 0.0f, ALPHAVALUE);
00177           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00178                   glColor4f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue, ALPHAVALUE);
00179           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00180                   glColor4f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue, ALPHAVALUE);
00181           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00182                   glColor4f(SELECT.red, SELECT.green, SELECT.blue, ALPHAVALUE);
00183           else
```

```
00184                   glColor4f(NOPION.red, NOPION.green, NOPION.blue, ALPHAVALUE);
00185 #else
00186           if (ARRETE)
00187                   glColor3f(0.0f, 0.0f, 0.0f);
00188           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00189                   glColor3f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue);
00190           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00191                   glColor3f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue);
00192           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00193                   glColor3f(SELECT.red, SELECT.green, SELECT.blue);
00194           else
00195                   glColor3f(NOPION.red, NOPION.green, NOPION.blue);
00196 #endif
00197           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + (k + RATIO1) *
zcase);
00198           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + (k +
RATIO1) * zcase);
00199           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + k *
zcase);
00200           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + k * zcase);
00201       glEnd();
00202       /* face 5 du cube */
00203       glBegin(state);
00204 #ifdef WITHALPHA
00205           if (ARRETE)
00206                   glColor4f(0.0f, 0.0f, 0.0f, ALPHAVALUE);
00207           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00208                   glColor4f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue, ALPHAVALUE);
00209           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00210                   glColor4f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue, ALPHAVALUE);
00211           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00212                   glColor4f(SELECT.red, SELECT.green, SELECT.blue, ALPHAVALUE);
00213           else
00214                   glColor4f(NOPION.red, NOPION.green, NOPION.blue, ALPHAVALUE);
00215 #else
00216           if (ARRETE)
00217                   glColor3f(0.0f, 0.0f, 0.0f);
00218           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00219                   glColor3f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue);
00220           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00221                   glColor3f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue);
00222           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00223                   glColor3f(SELECT.red, SELECT.green, SELECT.blue);
00224           else
00225                   glColor3f(NOPION.red, NOPION.green, NOPION.blue);
00226 #endif
00227           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + k * zcase);
00228           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + (k + RATIO1) * zcase);
00229           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + (k + RATIO1) *
zcase);
00230           glVertex3f(-1 + ECARTBORD + i * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + k * zcase);
00231       glEnd();
00232       /* face 6 du cube */
00233       glBegin(state);
00234 #ifdef WITHALPHA
00235           if (ARRETE)
```

```
00236                   glColor4f(0.0f, 0.0f, 0.0f, ALPHAVALUE);
00237           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00238                   glColor4f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue, ALPHAVALUE);
00239           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00240                   glColor4f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue, ALPHAVALUE);
00241           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00242                   glColor4f(SELECT.red, SELECT.green, SELECT.blue, ALPHAVALUE);
00243           else
00244                   glColor4f(NOPION.red, NOPION.green, NOPION.blue, ALPHAVALUE);
00245 #else
00246           if (ARRETE)
00247                   glColor3f(0.0f, 0.0f, 0.0f);
00248           else if (ContenuTab(plateau, i, j, k) == joueur1.id)
00249                   glColor3f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue);
00250           else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00251                   glColor3f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue);
00252           else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00253                   glColor3f(SELECT.red, SELECT.green, SELECT.blue);
00254           else
00255                   glColor3f(NOPION.red, NOPION.green, NOPION.blue);
00256 #endif
00257           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + k * zcase);
00258           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + j * ycase, -1 + ECARTBORD + (k + RATIO1) *
zcase);
00259           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + (k +
RATIO1) * zcase);
00260           glVertex3f(-1 + ECARTBORD + (i + RATIO1) * xcase, -1 + ECARTBORD + (j + RATIO1) * ycase, -1 + ECARTBORD + k *
zcase);
00261       glEnd();
00262       return;
00263 }
```

18) displayS2_opengl.c

```
00001 #include "global.h"
00002
00003 extern int subWindow2;
00004 extern int w1, h1;
00005 extern t_joueur joueur1, joueur2,* joueur;
00006 extern tplat plateau;
00007 extern char WINNER;
00008
00009 /*****************************************
00010  *RenderScenesw3:                                *
00011  *     On affiche les differents plans du cube *
00012  *****************************************/
00013 void RenderScenesw2(void)
00014 {
00015       unsigned char i, j, k;
00016       char buffer[SIZEBUFFER];
00017       int font;
00018       COORD place;
00019
00020       glutSetWindow(subWindow2);
00021       glClear(GL_COLOR_BUFFER_BIT);
00022       font = (int) GLUT_BITMAP_HELVETICA_18;
00023       /* Appel de la fonction de placement du pion de l'IA si necessaire */
```

```
00024       if (joueur->id == IANUMBER && !WINNER)
00025       {
00026             place = FindCoordIA(plateau, joueur, (joueur->id == IANUMBER) ? &joueur2 : &joueur1);
00027             Game(plateau, place, &joueur1, &joueur2);
00028       }
00029       /* Affichage des 3 plans du morpion */
00030       for (k = 0; k < NBZ; k++)
00031       {
00032             glPushMatrix();
00033             glLoadIdentity();
00034             /* On se met en 2D avec l'origine du repere en haut à gauche */
00035             gluOrtho2D(0, w1, 0, h1 / 2);
00036             glScalef(1, -1, 1);
00037             glTranslatef(0, -h1 / 2, 0);
00038             for (j = 0; j < NBY; j++)
00039             {
00040                   for (i = 0; i < NBX; i++)
00041                   {
00042
00043                         glBegin(GL_QUADS);
00044                               if (ContenuTab(plateau, i, j, k) == joueur1.id)
00045                                     glColor3f(joueur1.couleur.red, joueur1.couleur.green, joueur1.couleur.blue);
00046                               else if (ContenuTab(plateau, i, j, k) == joueur2.id)
00047                                     glColor3f(joueur2.couleur.red, joueur2.couleur.green, joueur2.couleur.blue);
00048                               else if (ContenuTab(plateau, i, j, k) == CASESELECT)
00049                                     glColor3f(SELECT.red, SELECT.green, SELECT.blue);
00050                               else
00051                                     glColor3f(NOPION.red, NOPION.green, NOPION.blue);
00052                               glVertex2f((k + 1) * ECARTTAB + i * XCASE2 + k * XTAB, ECARTTAB + j * YCASE2);
00053                               glVertex2f((k + 1) * ECARTTAB + (i + RATIO) * XCASE2 + k * XTAB, ECARTTAB + j * YCASE2);
00054                               glVertex2f((k + 1) * ECARTTAB + (i + RATIO) * XCASE2 + k * XTAB, ECARTTAB + (j + RATIO) *
YCASE2);
00055                               glVertex2f((k + 1) * ECARTTAB + i * XCASE2 + k * XTAB, ECARTTAB + (j + RATIO) * YCASE2);
00056                         glEnd();
00057                   }
00058             }
00059             /* On affiche du texte sous les tableaux pour indiquer de quel tableau il s'agit */
00060             glPopMatrix();
00061             glColor3f(TEXTE.red, TEXTE.green, TEXTE.blue);
00062             setOrthographicProjection();
00063             glPushMatrix();
00064             glLoadIdentity();
00065             sprintf(buffer, "Profondeur %d", k);
00066             RenderBitmapString(POSTEXTEX2, POSTEXTEY2, (void *)font, buffer);
00067             glPopMatrix();
00068             resetPerspectiveProjection();
00069       }
00070       glFlush();
00071       glutSwapBuffers();
00072       return;
00073 }
```

19) keyboard_opengl.c

```
00001 #include "global.h"
00002
```

```
00003 extern char WINNER;
00004 extern int mainWindow;
00005 extern t_joueur *joueur, joueur1, joueur2;
00006 extern char* plateau;
00007
00008 /*****************************************************************************
00009  *processSpecialKeys:                                                        *
00010  *      Ce qui se produit si on appuie sur des touches spéciales du clavier     *
00011  *****************************************************************************/
00012 void processSpecialKeys(int touche, int x, int y)
00013 {
00014       /* ALT + F4: on quitte le jeu */
00015       if (touche == GLUT_KEY_F4 && glutGetModifiers() == GLUT_ACTIVE_ALT)
00016       {
00017             glutDestroyWindow(mainWindow); exit(0);
00018       }
00019       return;
00020 }
00021
00022 /*****************************************************************************
00023  *ProcessNormalKeys:                                                         *
00024  *      Ce qui se produit si on appuie sur les touches classiques du clavier    *
00025  *****************************************************************************/
00026 void ProcessNormalKeys(unsigned char key, int x, int y)
00027 {
00028       key = tolower(key);
00029       switch (key)
00030       {
00031       /* S ou s: On sauvegarde le jeu */
00032       case 's':
00033             glutDestroyWindow(mainWindow);
00034             SaveGame(plateau, joueur, (joueur == &joueur1) ? &joueur2 : &joueur1);
00035             InitWindow();
00036             glutPostRedisplay();
00037             break;
00038       case 'q':
00039             glutDestroyWindow(mainWindow);
00040             mainWindow = 0; exit(0);
00041             break;
00042       /* N'importe quelle touche si il y a un vainqueur: On quitte le jeu */
00043       default:
00044             if (WINNER)
00045             {
00046                   glutDestroyWindow(mainWindow);
00047                   /* Dans l'éventualité ou l'on quitte le jeu, on met l'identifiant de la fenetre à 0 */
00048                   mainWindow = 0;
00049             }
00050             break;
00051       }
00052       return;
00053 }
```

20) mouse_opengl.c

```
00001 #include "global.h"
```

```
00002
00003 /* Pour faire tourner le cube de la sous fenetre 1 */
00004 extern int w1, h1;
00005 extern int subWindow1;
00006 int anglex = 6, angley = 10, xold, yold, x, y;
00007 static char presse = 0;
00008 /* Pour selectionner une case dans le tableau de façon temporaire */
00009 static int a, b, c; /* Les coord de la case selectionnee */
00010 extern char ISSELECT, WINNER;
00011 extern t_joueur joueur1, joueur2;
00012 extern tplat plateau;
00013 /*****************************************************************************************
00014  *Mouse1:                                                                                *
00015  *      Le bouton gauche de la souris est-il appuyé et sommes nous dans la sous fenetre 1       *
00016  *****************************************************************************************/
00017 void Mouse1(int button, int state,int x,int y)
00018 {
00019       /* si on appuie sur le bouton gauche */
00020       if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
00021       {
00022             presse = 1;     /* le booleen presse passe a 1 (vrai) */
00023             xold = x;       /* on sauvegarde la position de la souris */
00024             yold = y;
00025       }
00026       /* si on relache le bouton gauche */
00027       if (button == GLUT_LEFT_BUTTON && state == GLUT_UP)
00028       presse = 0;             /* le booleen presse passe a 0 (faux) */
00029       return;
00030 }
00031
00032 /*************************************************************
00033  *MouseMotion1:                                                  *
00034  *      Si le bouton gauche est presse, on travaille sur les angles     *
00035  *      de rotation du cube en fonction du mouvement de la souris       *
00036  *************************************************************/
00037 void MouseMotion1(int x,int y)
00038 {
00039       if (presse)           /* si le bouton gauche est presse */
00040       {
00041             /* on modifie les angles de rotation de l'objet
00042             en fonction de la position actuelle de la souris et de la derniere position sauvegardee */
00043             anglex += (x - xold);
00044             angley += (y - yold);
00045             glutSetWindow(subWindow1);
00046             glutPostRedisplay();    /* on demande un rafraichissement de l'affichage */
00047       }
00048       xold = x;       /* sauvegarde des valeurs courante de le position de la souris */
00049       yold = y;
00050       return;
00051 }
00052
00053 /*********************************************************
00054  *Mouse2:                                                    *
00055  *      Récupere la case sur laquelle on vient de cliquer            *
```

```
00056 *     et met un pion dessus (en fonction du clic effectue)    *
00057 ************************************************************/
00058 void Mouse2(int button, int state, int x, int y)
00059 {
00060       unsigned char i, j, k, d, e, f;
00061       unsigned char DONE = FALSE;
00062       COORD select;
00063
00064       if (y > ECARTTAB && x > ECARTTAB && !WINNER)
00065       {
00066             /* On joue sur la case selectionnee */
00067             if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
00068             {
00069                   if (ISSELECT)
00070                   {
00071                         ISSELECT = FALSE;
00072                         ContenuTab(plateau, a, b, c) = PASDEPION;
00073                   }
00074                   /* On recherche les coordonnees suivant y (ce sont les plus simples! */
00075                   for (j = 0; j < NBY; j++)
00076                         if (y < ECARTTAB + (j + RATIO) * YCASE2)
00077                         {
00078                               e = j;
00079                               break;
00080                         }
00081                   for (k = 0; k < NBZ; k++)
00082                   {
00083                         if (DONE == TRUE)
00084                               break;
00085                         for (i = 0; i < NBX; i++)
00086                               if (x > (k + 1) * ECARTTAB + i * XCASE2 + k * XTAB && x < (k + 1) * ECARTTAB + (i + RATIO) *
XCASE2 + k * XTAB)
00087                               {
00088                                     d = i;
00089                                     f = k;
00090                                     DONE = TRUE;
00091                                     break;
00092                               }
00093                   }
00094                   if (DONE && (ContenuTab(plateau, d, e, f) == PASDEPION || ContenuTab(plateau, d, e, f) == CASESELECT))
00095                   {
00096
00097                         select.x = d; select.y = e; select.z = f;
00098                         Game(plateau, select, &joueur1, &joueur2);
00099                   }
00100
00101             }
00102             /* Pour visualiser une case sans pour autant jouer dessus! */
00103             if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
00104             {
00105                   if (ISSELECT)
00106                   {
00107                         ISSELECT = FALSE;
00108                         ContenuTab(plateau, a, b, c) = PASDEPION;
```

```
00109                 }
00110                 /* On recherche les coordonnees suivant y (ce sont les plus simples! */
00111                 for (j = 0; j < NBY; j++)
00112                     if (y < ECARTTAB + (j + RATIO) * YCASE2)
00113                     {
00114                         b = j;
00115                         break;
00116                     }
00117                 for (k = 0; k < NBZ; k++)
00118                 {
00119                     if (DONE == TRUE)
00120                         break;
00121                     for (i = 0; i < NBX; i++)
00122                         if (x > (k + 1) * ECARTTAB + i * XCASE2 + k * XTAB && x < (k + 1) * ECARTTAB + (i + RATIO) *
XCASE2 + k * XTAB)
00123                         {
00124                             a = i;
00125                             c = k;
00126                             DONE = TRUE;
00127                             break;
00128                         }
00129                 }
00130                 if (DONE && ContenuTab(plateau, a, b, c) == PASDEPION)
00131                 {
00132                     ISSELECT = TRUE;
00133                     ContenuTab(plateau, a, b, c) = CASESELECT;
00134                 }
00135             }
00136         }
00137         return;
00138 }
```

21) font_opengl.c

```
00001 #include "global.h"
00002
00003 extern int w1, h1;
00004
00005 /*****************************************************************************
00006  * setOrthographicProjection:
00007  *      Bascule la matrice, la sauvegarde et l'altère pour l'affichage de texte
00008  *****************************************************************************/
00009 void setOrthographicProjection(void)
00010 {
00011     glMatrixMode(GL_PROJECTION); /* On se met en mode projection */
00012     glPushMatrix(); /* Sans oublier de sauvegarder l'ancienne matrice */
00013     glLoadIdentity(); /* On reset la matrice */
00014     gluOrtho2D(0, w1, 0, h1 / 2); /* On se met en projection 3D */
00015     glScalef(1, -1, 1); /* Inversion de l'axe des Y */
00016     glTranslatef(0, -h1 / 2, 0); /* Origine en haut à gauche */
00017     glMatrixMode(GL_MODELVIEW);
00018 }
00019
00020 /*****************************************************************************
00021  * resetPerspectiveProjection:
00022  *      Restaure la matrice précedente
```

```
00023 ******************************************************************************/
00024 void resetPerspectiveProjection(void)
00025 {
00026       glMatrixMode(GL_PROJECTION);
00027       glPopMatrix();
00028       glMatrixMode(GL_MODELVIEW);
00029 }
00030
00031 /*********************************************************************
00032 *RenderBitmapString:                                                 *
00033 *      Affiche à l'écran la chaine de caractère passée en paramètre    *
00034 *********************************************************************/
00035 void RenderBitmapString(float x, float y, void *font,char *string)
00036 {
00037       char *c;
00038
00039       /* Se positionne a l'emplacement specifie pour ecrire le texte */
00040       glRasterPos2f(x, y);
00041       /* Ecrit les caractères un par un */
00042       for (c=string; *c != '\0'; c++)
00043             glutBitmapCharacter(font, *c);
00044       return;
00045 }
```

22) client.c

```
00001 #include "network.h"
00002
00003 /*****************************************************************************
00004 * InitClient
00005 *      Prépare un socket pour la connexion du client
00006 *****************************************************************************/
00007 char InitClient(int* fd)
00008 {
00009       extern int errno;
00010       *fd = socket(PF_INET, SOCK_STREAM, 6);
00011       /* ipv4 flux TCP */
00012       if(*fd==-1) { return FALSE; }
00013       puts("Socket créé");
00014       return TRUE;
00015 }
00016
00017 /*****************************************************************************
00018 * Connect
00019 *      Prépare puis établit la connexion avec l'hôte distante
00020 *****************************************************************************/
00021 char Connect(int* fd, char* RemoteIP)
00022 {
00023       struct sockaddr_in remote;
00024       remote.sin_family = AF_INET; /* Type internet */
00025       remote.sin_addr.s_addr = INADDR_ANY; /* toute interface */
00026       remote.sin_port = htons(9999); /* port */
00027       if(inet_pton(AF_INET,RemoteIP,&remote.sin_addr)<=0)
00028             return FALSE;
00029       if(-1 == connect(*fd, (struct sockaddr *) &remote,sizeof(remote)))
```

```
00030       {
00031             printf("Impossible de se connecter : %s\n",
00032                         strerror(errno));
00033             return FALSE;
00034       }
00035       puts("Socket connectée");
00036       return TRUE;
00037 }
00038
00039
00040 /*****************************************************************************
00041  * StopClient
00042  *      Interrompt la connexion du client
00043  *****************************************************************************/
00044 void StopClient(int* fd)
00045 {
00046       shutdown(*fd,2);
00047 }
```

23) server.c

```
00001 #include "network.h"
00002 int fd; /* Socket permanent du serveur */
00003
00004 /*****************************************************************************
00005  * RunServer:
00006  *      Créé un socket et place le programme en attente d'une connexion
00007  *****************************************************************************/
00008 char RunServer(int* newfd)
00009 {
00010       struct sockaddr_in server, remote;
00011       int remotesize=sizeof(struct sockaddr_in);
00012
00013       fd = socket(PF_INET, SOCK_STREAM, 6);
00014       /* ipv4 flux TCP */
00015       if(fd==-1)
00016       {
00017             printf("Impossible de créer le socket : %s\n",
00018             strerror(errno));
00019             return FALSE;
00020       }
00021
00022       puts("Socket créé");
00023
00024       server.sin_family = AF_INET; /* Type de famille d'adresse : internet */
00025       server.sin_addr.s_addr = INADDR_ANY; /* toute interface */
00026       server.sin_port = htons(9999); /* port */
00027
00028       {
00029             unsigned char retry = 0;
00030             do{
00031                   if(-1 == bind(fd, (struct sockaddr *) &server,sizeof(server)))
00032                   {
00033                         printf("Impossible d'attacher le socket : %s\n",
00034                               strerror(errno));
```

```
00035                       if(retry>MAXNETRETRY)
00036                           return FALSE;
00037                       else
00038                           sleep(3);
00039                   }
00040                   else
00041                   {
00042                       puts("Socket attachée");
00043                       break;
00044                   }
00045               }while(1);
00046       }
00047
00048
00049       if(-1 == listen(fd,1))
00050       {
00051           return FALSE;
00052       }
00053       else    puts("Socket en écoute");
00054       *newfd = accept(fd, (struct sockaddr *)&remote, (socklen_t *) &remotesize);
00055       if(*newfd==-1)
00056       {
00057           printf("Connexion échouée : %s\n",
00058               strerror(errno));
00059           return FALSE;
00060
00061       }
00062       printf("Connexion établie avec %s\n",inet_ntoa(remote.sin_addr));
00063       return TRUE;
00064 }
00065
00066
00067 /*****************************************************************************
00068  * StopServer:
00069  *      Arrêt du serveur par fermeture du socket
00070  *****************************************************************************/
00071 void StopServer(int *newfd)
00072 {
00073       puts("Arret du serveur, le client s'est déconnecté");
00074       close(*newfd);
00075       close(fd);
00076 }
```

24) network.c

```
00001 #include "network.h"
00002
00003 /*****************************************************************************
00004  * NetSend:
00005  *      Envoie un message sur le socket passé en paramètre
00006  *****************************************************************************/
00007 char NetSend(int* newfd, char* buffer, unsigned int bufsize)
00008 {
00009       if(send(*newfd, buffer, bufsize,0)!=-1)
00010       {
```

```
00011             /*printf("Message envoyé: '%s'\n", buffer);*/
00012             return TRUE;
00013       }
00014       else    return FALSE;
00015 }
00016
00017 /*****************************************************************************
00018  * NetReceive
00019  *      Attend l'arrivée d'un message à partir du socket passé en paramètre et récupère son contenu
00020  *****************************************************************************/
00021 char NetReceive(int* newfd, char* buffer, unsigned int bufsize)
00022 {
00023       int car;
00024       car=recv(*newfd, buffer, bufsize,0);
00025       buffer[car]='\0';
00026
00027             /*printf("Message reçu: '%s' (%d)\n", buffer,car);*/
00028       return (car==0)? FALSE : TRUE;
00029 }
```